On the Numerics of Estimating Generalized Hyperbolic Distributions

A Master Thesis Presented

by

Wang, Congcong

(189010)

to

Prof. Dr. Wolfgang Härdle

CASE - Center of Applied Statistics and Economics

Humboldt-Universität zu Berlin

in partial fulfillment of the requirements

for the degree of

Master of Science

Berlin, September 15, 2005

DECLARATION OF AUTHORSHIP

I hereby confirm that I have authored this master thesis independently and without use of others than the indicated resources. All passages, which are literally or in general matter taken out of publications or other resources, are marked as such.

Wang, Congcong

Berlin, September 15, 2005

ACKNOWLEDGMENT

I would like to thank Professor Dr.Wolfgang Härdle for giving me the opportunity and motivation to write this thesis.

I'm especially indebted to Ying Chen for her excellent guidance all the time.

Furthermore, I'm also grateful to my family and friends, without their support it would be impossible to finish this work.

ABSTRACT

An important empirical fact in financial market is that return distributions are often skewed and heavy-tailed. This paper employs maximum likelihood estimation to estimate the five parameters of generalized hyperbolic distribution, a highly flexible heavy-tailed distribution. The estimation utilizes Powell's methods in multidimensions and the performance of estimation is measured by simulation studies. Application to the financial market provides us with estimates of return distribution of some financial assets.

Key words and Phrases: Generalized Hyperbolic distribution, Maximum Likelihood Estimation, Powell's Methods in Multidimensions.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This thesis focuses on a particular heavy-tailed distribution: generalized hyperbolic (GH) distribution. The aim is to overview GH distributions and to estimate the five parameters of GH distributions in the financial environment.

Heavy-tailed distributions were first introduced by the Italian-born Swiss economist Pareto (1896) and extensively studied by Paul Lévy. Although then these distributions were mainly studied theoretically, nowadays they have found many applications in areas as diverse as finance, medicine, seismology, structural engineering.

A distribution is called heavy-tailed if it has higher probability density in its tail areas compared with a normal distribution with the same mean μ and variance σ^2. Figure 1.1 demonstrates the differences of the pdf curves of a standard Gaussian distribution and a Cauchy distribution with location parameter $\mu = 0$ and scale parameter $\sigma = 1$. The graphic shows that the probability density of the Cauchy distribution is much higher than that of the Gaussian in the tail part, while in the area around the center, the probability density of the Cauchy distribution is much lower.

In terms of kurtosis, a heavy-tailed distribution has kurtosis greater than 3, which is called leptokurtic, in contrast to mesokurtic distribution (kurtosis = 3) and platykurtic distribution (kurtosis < 3).

An important empirical fact in financial market is that return distributions are often skewed and have heavier tails than the normal distribution. Risk management based on normal assumptions may therefore lead to underestimation of the risk. Researchers have tried to fix this by offering other classes of distributions, first the stable Paretian class and more recently the generalized hyperbolic class.

One of the reasons, which make the GH distributions so popular, is that its five parameters are flexible enough to fit many different data sets well and make GH distributions potentially useful in many different contexts. As a striking feature of GH distribution, it embraces many subclasses and limiting distributions, e.g. hyperbolic, normal inverse Gaussian, Student-t and normal distributions. All of them have been used to model financial

returns.

In recent years normal inverse Gaussian (NIG) distribution, a subclass of GH distribution, has been successfully fitted to returns in financial time series by many researchers; see Eberlein and Keller (1995), Prause (1997), Barndorff-Nielsen (1997), Prause (1999), Barndorff-Nielsen and Shephard (2001). This has opened an area, in which NIG distributions are used as building blocks to model the time dynamics of financial markets.

Since NIG distribution is a special case of GH distribution, in which one of the parameters of GH distribution, namely λ, is fixed to -1/2, estimation of NIG distribution is actually a four-parameter estimation. It is rather interesting to further exploit GH distributions, which means to include the parameter λ in the estimation, a step from four-parameter to five-parameter estimation.

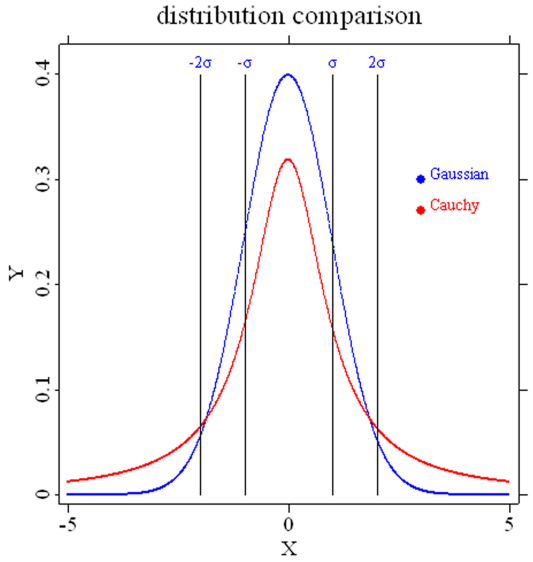

Fig. 1.1: Comparison between the pdf curves of a standard Gaussian (blue) and a Cauchy distribution (red) with location parameter 0 and scale parameter 1. MVAcg.xpl

The work is divided into four parts. The first section starts with definition of GH distributions, the subclasses and limiting distributions of GH distributions, as well as the normal variance-mean mixture presentation. Their properties are also examined in the first part: moment generating function and characteristic function. The second section presents the maximum likelihood estimation and numerical algorithms used to estimate parameters, including Golden Section search and parabolic interpolation, on which

Powell's method in multi-dimensions are based. In the third part, various original parameter sets are employed in simulation studies to measure the performance of estimation under different situations. Finally, part 4 uses real data to estimate GH distribution density and compares the results with those of nonparametric method.

All results and generated codes using XploRe and C language are gathered at the appendix. The bibliography contains classical references on GH distributions, where deeper computational and mathematical treatment can be found.

2. GENERALIZED HYPERBOLIC DISTRIBUTIONS

The generalized hyperbolic distribution was introduced by Barndorff-Nielsen (1977) for modeling grain size distributions of wind blown sands. The original paper focused on the special case of the hyperbolic distribution. The name of the distributions is derived from the fact that its log-density forms a hyperbola, while the log-density of the normal distribution is a parabola.

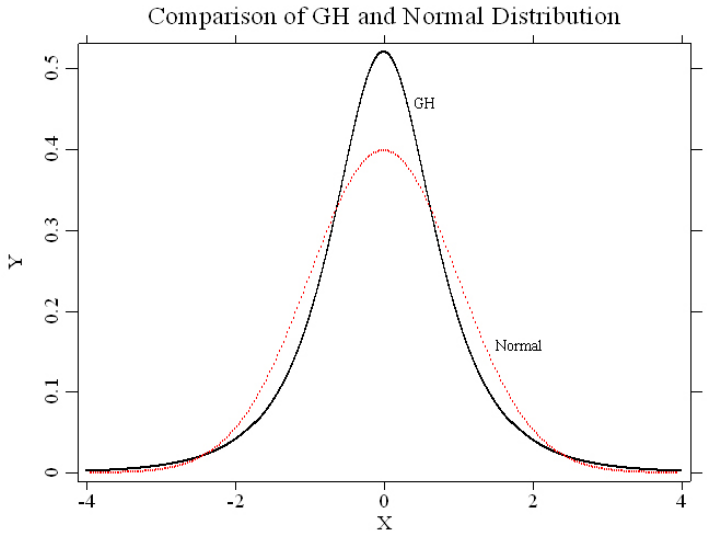

Fig. 2.1: Comparison between pdf curves of GH(-0.5,1,0,1,0) and normal(0,1) distribution. GHvsN.xpl

GH distributions embrace many subclasses and limiting distributions, e.g. hyperbolic, normal inverse Gaussian, Student-t and normal distributions, which will be discussed in detail in Section 2.3. Being a normal variance-mean mixture, GH distributions possess heavy tails, i.e. the kurtosis is higher than that of normal distribution. Hence it provides the possibility of modeling the well-known heavy tails of return distributions for most financial assets. Figure 2.1 compares the pdf curves of a GH distribution (black) and a normal distribution (red dotted). Both distributions have mean 0 and variance 1.

2.1 Definition and Parameterization

2.1.1 Probability Density Function and Parameterization

We denote the one-dimensional generalized hyperbolic (GH) distribution by $GH(x;\lambda,\alpha,\beta,\delta,\mu)$ for $x \in \mathbb{R}$. It then can be characterized via its probability density:

$$\begin{aligned} f_{GH}(x;\lambda,\alpha,\beta,\delta,\mu) &= a(\lambda,\alpha,\beta,\delta,\mu)\big\{\delta^2+(x-\mu)^2\big\}^{(\lambda-\frac{1}{2})/2} exp\{\beta(x-\mu)\} \\ &\quad \times K_{\lambda-\frac{1}{2}}\big(\alpha\sqrt{\delta^2+(x-\mu)^2}\big), \end{aligned} \tag{2.1}$$

with the norming constant

$$a(\lambda,\alpha,\beta,\delta,\mu) = \frac{\big(\alpha^2-\beta^2\big)^{\frac{\lambda}{2}}}{\sqrt{2\pi}\alpha^{\lambda-\frac{1}{2}}\delta^{\lambda}K_{\lambda}\big(\delta\sqrt{\alpha^2-\beta^2}\big)}, \tag{2.2}$$

where K_λ is a modified Bessel function of the third kind with index λ:

$$K_{\lambda}(x) = \frac{1}{2}\int_0^{\infty} y^{\lambda-1} exp\Big\{-\frac{x}{2}(y+y^{-1})\Big\}dy, \tag{2.3}$$

which shows the strict positivity of K_λ on $\mathbb{R}>0$. The substitution $x := y^{-1}$ immediately gives $K_{-\lambda} = K_\lambda$, Furthermore, $K_\lambda(x)$ is obviously monotonically decreasing in x on $\mathbb{R}>0$.

Besides $\mu \in \mathbb{R}$, the values which parameters can take are:

$$\begin{aligned} &\delta \geq 0, |\beta| < \alpha, \qquad \text{if} \quad \lambda > 0 \\ &\delta > 0, |\beta| < \alpha, \qquad \text{if} \quad \lambda = 0 \\ &\delta > 0, |\beta| \leq \alpha, \qquad \text{if} \quad \lambda < 0. \end{aligned} \tag{2.4}$$

α and β are sometimes replaced by alternative parameterizations in the literature:

$$\rho = \frac{\beta}{\alpha}, \quad \zeta = \delta\sqrt{\alpha^2-\beta^2} \tag{2.5}$$

$$\chi = \rho\xi, \quad \xi = \frac{1}{\sqrt{1+\zeta}}. \tag{2.6}$$

$$\bar{\alpha} = \delta\alpha, \quad \bar{\beta} = \delta\beta \tag{2.7}$$

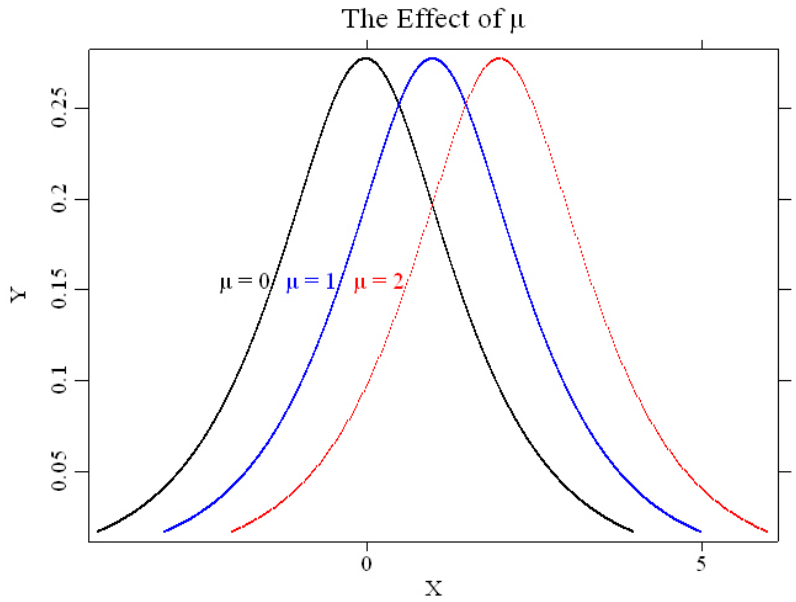

Fig. 2.2: Pdf curves of three GH distributions with $\lambda = 1.3, \alpha = 1, \beta = 0, \delta = 1$ and $\mu = 0$ (black), $\mu = 1$ (blue), $\mu = 2$ (red dotted) GHmu.xpl

For symmetric distributions $\beta = \bar{\beta} = \rho = \chi = 0$ holds.

Roughly speaking, μ is a location parameter, δ serves for scaling, β determines the skewness and α the shape. Increasing ξ or decreasing ζ or $\delta\alpha$ reflects an increase in the kurtosis. λ characterizes certain subclasses and considerably influences the size of the tails.

Figure 2.2 shows that the effect of location parameter μ is very clear: an increase in μ moves the pdf curves of GH distribution rightward horizontally.

The role played by scale parameter δ is demonstrated in Figure 2.3. With an increase in the value of δ, the pdf curve becomes flatter. At the same time, a raise on δ with α remaining constant decreases the kurtosis of the GH distributions. That's why we mentioned "roughly speaking" concerning the effects of the parameters in GH distributions, because they are multifold: one parameter can have an impact on different moments.

The versatility of parameter β is even more obvious in Figure 2.4. The main effect on the probability density is, that the pdf curve skews left when $\beta < 0$ and skews right when $\beta > 0$; the density is symmetric when $\beta = 0$. With larger value of absolute value of β, the skewness is more obvious. Besides the main effect, β also moves the pdf curve horizontally, which means it changes the mean. As demonstrated in Figure 2.4, the pdf curve moves rightward

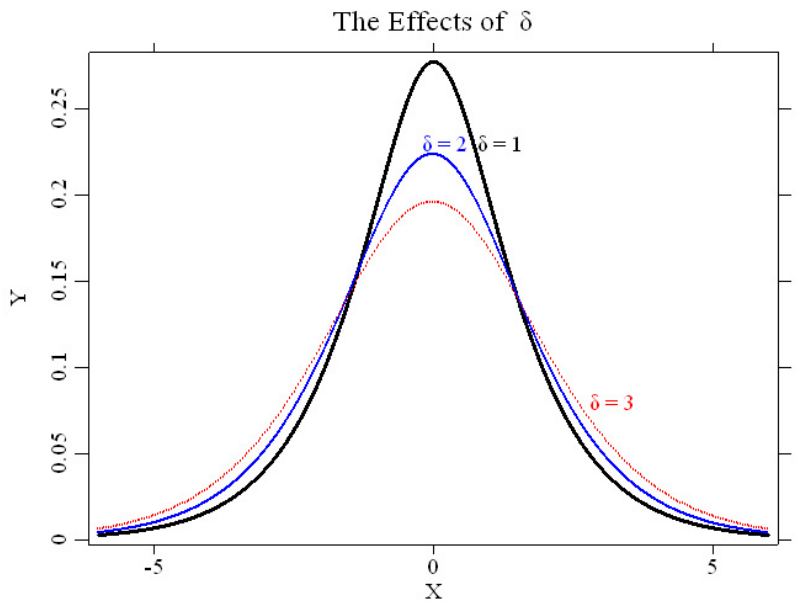

Fig. 2.3: Pdf curves of three GH distributions with $\lambda = 1.3, \alpha = 1, \beta = 0, \mu = 0$ and $\delta = 1$ (black), $\delta = 2$ (blue), $\delta = 3$ (red dotted) GHdelta.xpl

when β takes a positive value and moves leftward when β is negative. We will go to details when we introduce the moments of GH distributions.

Figure 2.5 illustrates the impact of parameter α. A decrease in α results in an increase in kurtosis, which peaks the pdf curve. At the same time, other parameters remaining constant, a decrease in α also forces the variance to increase, which in contrast flattens the curve, vice versa.

Finally let's take a look at the combined effects of $\alpha\delta$. As indicated before, an increase $\delta\alpha$ reflects a decrease in the kurtosis. In Figure 2.6, we gradually increase the value of α from 1 to 2.25, while decreasing the value of δ from 1.2 to 0.8, so that the products of α and δ increases from 1.2 to 1.5, and further to 1.8. The effects are obvious: the red dotted pdf curve, which has the largest $\alpha\delta$ value, has the fastest decaying speed in tail areas, implying it has the smallest kurtosis while the black curve has the slowest decaying speed.

With a different way parameterization, the role of the scale and location parameters δ respectively μ become more obvious. Barndorff-Nielsen and Stelzer (2005) represent the pdf of GH distributions as:

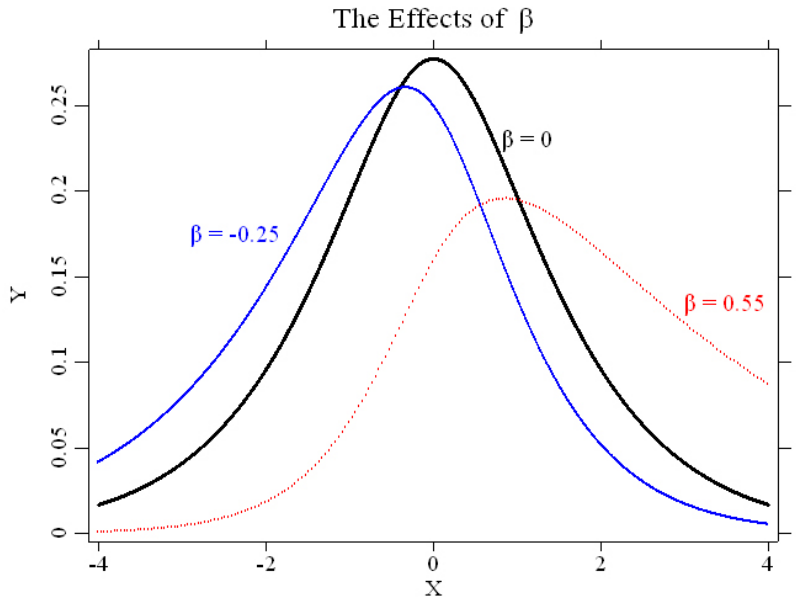

Fig. 2.4: Pdf curves of three GH distributions with $\lambda = 1.3, \alpha = 1, \delta = 1, \mu = 0$ and $\beta = 0$ (black), $\beta = -0.25$ (blue), $\beta = 0.55$ (red dotted) GHbeta.xpl

$$f_{GH}(x; \lambda, \alpha, \beta, \delta, \mu) = \frac{\bar{\gamma}^{\lambda} \bar{\alpha}^{1/2-\lambda}}{\sqrt{2\pi}\delta K_{\lambda}(\bar{\gamma})} \left\{ 1 + \frac{(x-\mu)^2}{\delta^2} \right\}^{\lambda/2-1/4} \times K_{\lambda-1/2}\left(\bar{\alpha}\sqrt{1 + \frac{(x-\mu)^2}{\delta^2}} \right) exp\{\beta(x-\mu)\} \quad (2.8)$$

where

$$\gamma = \sqrt{\alpha^2 - \beta^2}, \quad \bar{\alpha} = \delta\alpha, \\ \bar{\beta} = \delta\beta, \quad \bar{\gamma} = \delta\gamma. \quad (2.9)$$

For those, who are not familiar with details of GH distributions, the term $\left(\frac{x-\mu}{\delta}\right)$ is similar to normalization.

Suppose $X \sim GH(\lambda, \alpha, \beta, \delta, \mu)$, Blæsild (1981) proved that a linear transformation $Y = aX + b$ is again GH-distribution with parameters $\lambda^+ = \lambda, \alpha^+ = \alpha/|a|, \beta^+ = \beta/|a|, \delta^+ = \delta|a|$ and $\mu^+ = a\mu + b$, which means

$$\widetilde{X} = aX + b \sim GH(\lambda, \alpha/|a|, \beta/|a|, \delta|a|, a\mu + b). \quad (2.10)$$

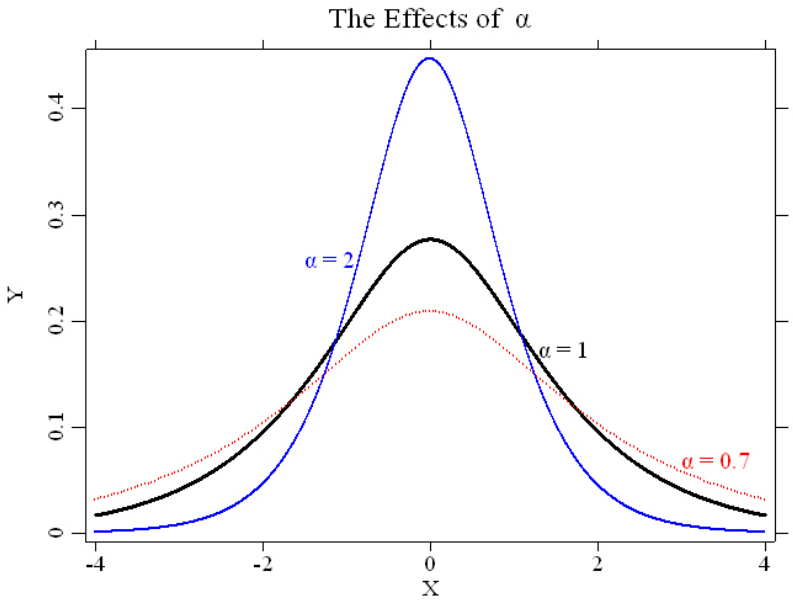

Fig. 2.5: Pdf curves of three GH distributions with $\lambda = 1.3, \beta = 0, \delta = 1, \mu = 0$ and $\alpha = 1$ (black), $\alpha = 2$ (blue), $\alpha = 0.7$ (red dotted) GHalpha.xpl

Since $\alpha^+\delta^+ = \alpha\delta$ and $\beta^+\delta^+ = \beta\delta$ holds, the term $\lambda, \bar{\alpha}$ and $\bar{\beta}$ are scale- and location-invariant parameters of the univariate GH distributions. The same holds for the other parameterizations (ζ, ρ) and (ξ, χ).

2.1.2 Representation as A Normal Variance-mean Mixture

When we work with GH distributions, it is sometimes more convenient to represent them in other forms. For example, GH distributions can be represented as a normal variance-mean mixture with normal distribution of mean $\xi = \mu + \beta\sigma^2$ and variance σ^2. The mixture includes generalized inverse Gaussian (GIG) distributions. The representation as mixture is very helpful in studying GH distributions, since GIG is often used to generate GH random variables. See rndgh.xpl.

Mixture modeling means to modeling a statistical distribution by a mixture (or weighted sum) of different distributions. With unrestricted choices of component density functions, it can approximate any continuous density to arbitrary accuracy, given sufficiently large number of component. The pdf

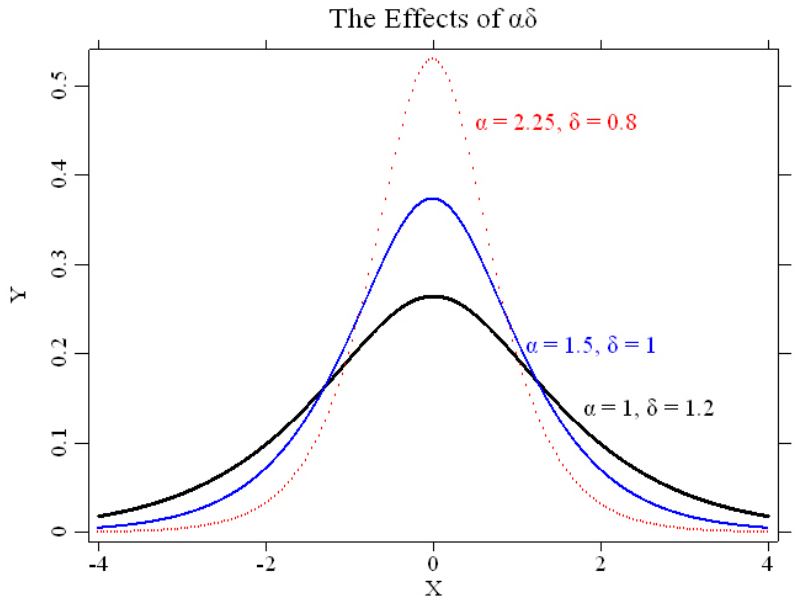

Fig. 2.6: Pdf curves of three GH distributions with $\lambda = 1.3, \beta = 0, \mu = 0$ and $\alpha = 1, \delta = 1.2$ (black), $\alpha = 1.5, \delta = 1$ (blue), $\alpha = 2.25, \delta = 0.8$ (red dotted curve)
GHad.xpl

of a mixture which consists of n distributions can be written as

$$f(x) = \sum_{l=1}^{n} w_l p_l(x), \tag{2.11}$$

under the constraints:

$$\begin{aligned} 0 \leq w_l &\leq 1 \\ \sum_{l=1}^{n} w_l &= 1 \\ \int p_l(x)dx &= 1, \end{aligned}$$

where $p_l(x)$ is the pdf of l'th distribution or l'th component density and w_l is called coefficient, which can be viewed as weight of l'th distribution in the mixture.

If we denote the generalized inverse Gaussian as $GIG(x; \lambda, \chi, \psi)$, GH distributions can be represented as:

$$f_{GH}(x; \lambda, \alpha, \beta, \delta, \mu) = \int_0^\infty N(x; \mu + \beta\omega, \omega) GIG(\omega; \lambda, \delta^2, \alpha^2 - \beta^2) d\omega, \tag{2.12}$$

where N is the normal density function with respect to mean $\mu + \beta\omega$ and variance ω. $GIG(x; \lambda, \chi, \psi)$ has the following probability density function:

$$f_{GIG}(x; \lambda, \chi, \psi) = \frac{(\psi/\chi)^{\lambda/2}}{2K_\lambda(\sqrt{\psi\chi})} e(x; \lambda, \chi, \psi), \tag{2.13}$$

with

$$e(x; \lambda, \chi, \psi) = x^{\lambda-1} exp\{-(1/2)(\chi x^{-1} + \psi x)\}, x > 0. \tag{2.14}$$

See Prause (1999). The domain of variation for the parameters is

$$\begin{array}{lll} \chi > 0, & \psi \geq 0 & \text{if} \quad \lambda < 0, \\ \chi > 0, & \psi > 0 & \text{if} \quad \lambda = 0, \\ \chi \geq 0, & \psi > 0 & \text{if} \quad \lambda > 0. \end{array}$$

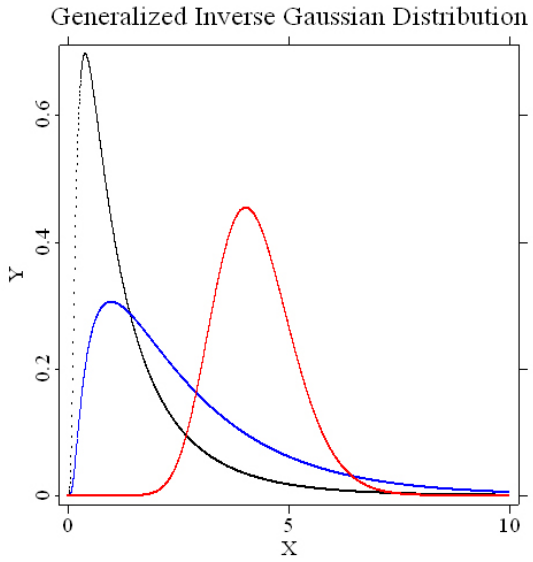

Fig. 2.7: The density function of the GIG distribution: $\lambda = 0, \chi = 1$(black), $\lambda = 1, \chi = 1$(blue) and $\lambda = 20, \chi = 10$(red). GHgig.xpl

With the parameterization of GH distributions given in equation 2.9, Barndorff-Nielsen and Stelzer (2005) write the GIG density function as:

$$\begin{aligned} f_{GIG}(x;\lambda,\delta,\gamma) &= \frac{(\gamma/\delta)^{\lambda}}{2K_{\lambda}(\delta\gamma)} x^{\lambda-1} exp\Big\{-\frac{1}{2}(\delta^2 x^{-1}+\gamma^2 x)\Big\} \\ &= \frac{\bar{\gamma}^{\lambda}}{2K_{\lambda}(\bar{\gamma})} \delta^{-2\lambda} x^{\lambda-1} exp\Big\{-\frac{1}{2}(\delta^2 x^{-1}+\bar{\gamma}^2\delta^{-2}x)\Big\} \end{aligned} \quad (2.15)$$

The density can also be reparameterized by setting $\xi = \sqrt{\chi\psi}$, a shape parameter, and $\eta = \sqrt{\chi/\psi}$, a scale parameter. When $\eta = 1$, $\xi = \chi = \psi$. See Atkinson (1982). Figure 2.7 demonstrates three members of generalized inverse Gaussian distribution with the same scale parameter $\eta = 1$ but different shape parameters.

2.2 Properties

2.2.1 Moment Generating Function

We assume that $\mu = 0$ for simplicity. Since

$$\begin{aligned} \int f_{GH}(x;\lambda,\alpha,\beta,\delta,0) &= 1, \\ \int \{\delta^2+x^2\}^{(\lambda-\frac{1}{2})/2} exp\{\beta x\} K_{\lambda-\frac{1}{2}}\big(\alpha\sqrt{\delta^2+x^2}\big)dx &= \frac{1}{a(\lambda,\alpha,\beta,\delta,0)}, \end{aligned}$$

the moment generating function of GH distributions with $|\beta+u| < \alpha$ is simply the ratio of the norming constants a defined in equation 2.2 corresponding to the parameters $(\lambda,\alpha,\beta,\delta)$ and $(\lambda,\alpha,\beta+\mu,\delta)$. See Prause (1999).

$$\begin{aligned} M_{GH}(u) &= \int exp\{ux\} f_{GH}(x;\lambda,\alpha,\beta,\delta,0)dx \\ &= a(\lambda,\alpha,\beta,\delta)\int exp\{ux\}(\delta^2+x^2)^{\frac{1}{2}\left(\lambda-\frac{1}{2}\right)} K_{\lambda-\frac{1}{2}}\big(\alpha\sqrt{\delta^2+x^2}\big) exp\{\beta x\}dx \\ &= \frac{a(\lambda,\alpha,\beta,\delta)}{a(\lambda,\alpha,\beta+\mu,\delta)} \\ &= \frac{(\alpha^2-\beta^2)^{\lambda/2}}{\sqrt{2\pi}\delta^{\lambda}\alpha^{\lambda-1/2}K_{\lambda}\big(\delta\sqrt{\alpha^2-\beta^2}\big)} \frac{\sqrt{2\pi}\delta^{\lambda}\alpha^{\lambda-1/2}K_{\lambda}\big(\delta\sqrt{\alpha^2-(\beta+u)^2}\big)}{\big\{\alpha^2-(\beta+u)^2\big\}^{\lambda/2}} \\ &= \left\{\frac{\alpha^2-\beta^2}{\alpha^2-(\beta+u)^2}\right\}^{\lambda/2} \frac{K_{\lambda}\big(\delta\sqrt{\alpha^2-(\beta+\mu)^2}\big)}{K_{\lambda}\big(\delta\sqrt{\alpha^2-\beta^2}\big)}. \end{aligned} \quad (2.16)$$

The moment generating function of the generalized hyperbolic distribution is given by:

$$M_{GH}(u) = exp\{u\mu\}\left\{\frac{\alpha^2-\beta^2}{\alpha^2-(\beta+u)^2}\right\}^{\lambda/2}\frac{K_\lambda\big(\delta\sqrt{\alpha^2-(\beta+u)^2}\big)}{K_\lambda\big(\delta\sqrt{\alpha^2-\beta^2}\big)}, \quad |\beta+u|<\alpha \tag{2.17}$$

in which the restriction $|\beta+u|<\alpha$ comes from the the domain of variation of the parameters in equations 2.4.

If $X \sim GH(x;\lambda,\alpha,\beta,\delta,\mu)$, Gut (1995) proved that all moments of X exist. Generalized hyperbolic distributions therefore possess moments of arbitrary order. In particular we take the first two derivatives of $M_{GH}(u)$ to find the mean and variance.

We assume $\mu = 0$ without loss of generality. Since $K'_\lambda(x) = -K_{\lambda+1}(x) + \frac{\lambda}{x}K_\lambda(x)$, Prause (1999) shows that,

$$\begin{aligned} M'_{GH}(u) &= \frac{(\alpha^2-\beta^2)^{\lambda/2}}{K_\lambda\big(\delta\sqrt{\alpha^2-\beta^2}\big)}\Big\{K_\lambda\big(\delta\sqrt{\alpha^2-(\beta+u)^2}\big)\big\{\alpha^2-(\beta+u)^2\big\}^{-\lambda/2}\Big\}' \\ &= \frac{(\alpha^2-\beta^2)^{\lambda/2}}{K_\lambda\big(\delta\sqrt{\alpha^2-\beta^2}\big)}\frac{K_{\lambda+1}\big(\delta\sqrt{\alpha^2-(\beta+u)^2}\big)}{\big\{\alpha^2-(\beta+u)^2\big\}^{(\lambda+1)/2}}. \end{aligned} \tag{2.18}$$

If we insert 0 for u we obtain

$$M'_{GH}(0) = \frac{\beta\delta K_{\lambda+1}\big(\delta\sqrt{\alpha^2-\beta^2}\big)}{\sqrt{\alpha^2-\beta^2}K_\lambda\big(\delta\sqrt{\alpha^2-\beta^2}\big)}. \tag{2.19}$$

Applying the definition of γ and $\bar{\gamma}$ in 2.9 gives the mean of generalized hyperbolic distribution:

$$\mathrm{E}[X] = \mu + \beta\frac{\delta K_{\lambda+1}(\bar{\gamma})}{\gamma K_\lambda(\bar{\gamma})}. \tag{2.20}$$

Taking the second derivative gives the variance:

$$\mathrm{Var}[X] = \delta^2\left[\frac{K_{\lambda+1}(\bar{\gamma})}{\bar{\gamma}K_\lambda(\bar{\gamma})} + \frac{\beta^2}{\gamma^2}\left\{\frac{K_{\lambda+2}(\bar{\gamma})}{K_\lambda(\bar{\gamma})} - \left(\frac{K_{\lambda+1}(\bar{\gamma})}{K_\lambda(\bar{\gamma})}\right)^2\right\}\right]. \tag{2.21}$$

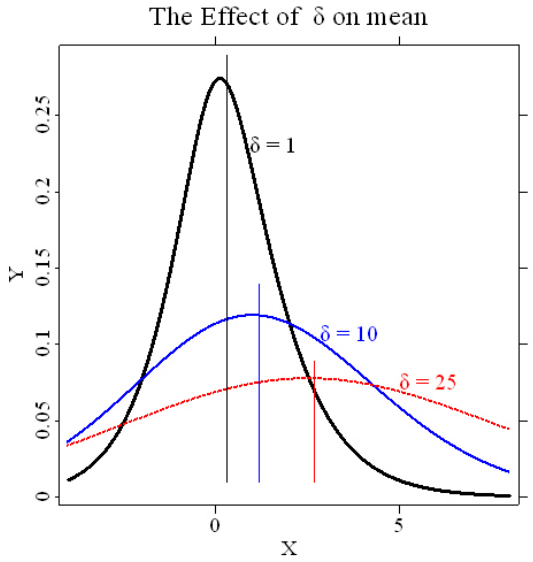

Fig. 2.8: Pdf curves of three GH distributions with $\lambda = 1.3, \alpha = 1, \beta = 0.1, \mu = 0$ and $\delta = 1$ (black), $\delta = 5$ (blue), $\delta = 15$ (red). `Mdelta.xpl`

2.2.2 Mean and Variance

Higher order moments can also be calculated because of the existence of all moments, but the expressions become more and more complicated. Clearly, both formulae for mean and variance are less complicated in the symmetric case, e.g. when we assume $\beta = 0$, recalling that β roughly speaking describes the skewness. The mean is simply μ under this condition.

The expression of mean of GH distributions in equation 2.20 shows that $\lambda, \alpha, \beta, \delta$ and μ all have effects on the mean. The impacts of μ and β have been illustrated by Figure 2.2 and Figure 2.4 respectively. With the parameter set in Figure 2.4, it is easy to obtain, that $\mathrm{E}[X] = 0$ for the black curve, $\mathrm{E}[X] = -0.8385$ for the blue curve and $\mathrm{E}[X] = 2.3971$ for the red dotted curve. See Appendix 7.2.3: ghmv.xpl. However the effect of α, δ and λ is not developed, since β takes the value of 0 in Figure 2.3 and Figure 2.5, therefore the second term of equation 2.20 does not influence the mean. By setting $\beta = 0.1$, Figure 2.8 and 2.9 demonstrate the impacts of δ and λ on the first moment of GH distribution. It is easy to discern, that the mean of GH distribution increases in δ and also in λ under given parameters setting. In the graphics, the curves move rightward with the increase of δ and λ.

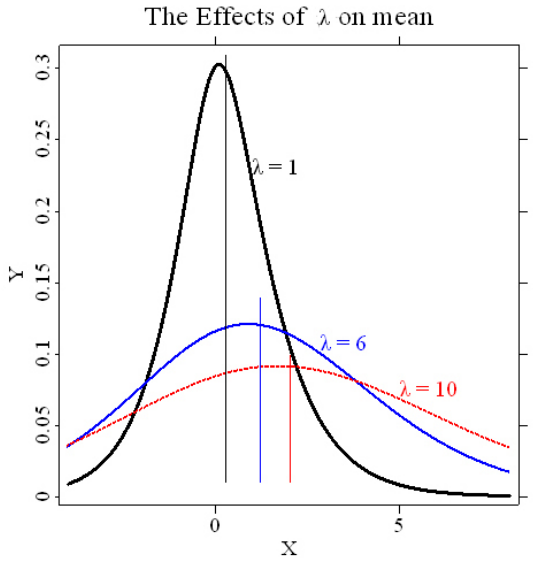

Fig. 2.9: Pdf curves of three GH distributions with $\alpha = 1, \beta = 0.1, \delta = 1, \mu = 0$ and $\lambda = 1$ (black), $\lambda = 6$ (blue), $\lambda = 10$ (red). Mlambda.xpl

2.2.3 Characteristic Function

Prause (1999) showed, that the radius of convergence of the moment generating function M_{GH} around zero is $\alpha - \beta$, and with Gut (1995) Theorem III 3.3 the moment generating function M_{GH} is a real analytic, i.e. it can be expanded in a power series around zero.

The characteristic function of the generalized hyperbolic distribution is given by

$$\varphi_{GH}(u) = exp\{i\mu u\}\left\{\frac{\alpha^2-\beta^2}{\alpha^2-(\beta+iu)^2}\right\}^{\lambda/2}\frac{K_\lambda\left(\delta\sqrt{\alpha^2-(\beta+iu)^2}\right)}{K_\lambda\left(\delta\sqrt{\alpha^2-\beta^2}\right)}. \quad (2.22)$$

2.3 Subclasses and Limiting Distributions

With specific values of λ, different subclasses are identified. For $\lambda = 1$ we obtain hyperbolic (HYP) distributions and for $\lambda = -1/2$ we get the normal inverse Gaussian (NIG) distributions.

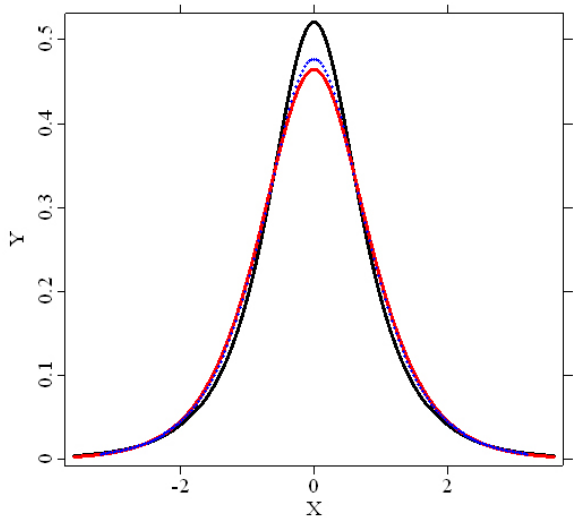

Fig. 2.10: Comparison between pdf curves of GH(0.5,1.615,0,1,0), NIG(-0.5,1,0,1,0) and HYP(1,1.875,0,1,0) distributions. `ghnighyp.xpl`

When $\lambda \in 1/2\mathbb{Z}$, the Bessel function K_λ can be simplified to

$$K_{n+1/2}(x) = \sqrt{\frac{\pi}{2}} x^{-1/2} exp\{-x\} \left(1 + \sum_{i=1}^{n} \frac{(n+i)!}{(n-i)!i!} (2x)^{-i} \right), \qquad (2.23)$$

for $\lambda = n+1/2$, $n = 0, 1, 2, ...$ Since $K_\lambda(x) = K_{-\lambda}(x)$, we obtain $K_{-1/2}(x) = K_{1/2} = \sqrt{\pi/2} x^{-1/2} exp\{-x\}$. This allows simpler expressions for probability density functions of HYP distributions and NIG distributions.

2.3.1 Hyperbolic Distributions

The HYP distribution has the following pdf:

$$f_{HYP}(x; \alpha, \beta, \delta, \mu) = \frac{\sqrt{\alpha^2 - \beta^2}}{2\alpha\delta K_1(\delta\sqrt{\alpha^2 - \beta^2})} exp\{-\alpha\sqrt{\delta^2 + (x-\mu)^2} + \beta(x-\mu)\} \qquad (2.24)$$

The focus of original paper by Barndorff-Nielsen (1977) was on the subclass of the HYP distributions. Barndorff-Nielsen (1982) then employed the three-dimensional HYP distributions in relativistic statistical Physics.

The application in particle size distributions of sand is further discussed by Barndorff-Nielsen, Blæsild, Jensen and Sørensen (1983), Barndorff-Nielsen, Blæsild, Jensen and Sørensen (1985). Barndorff-Nielsen and Christiansen (1988), Hartmann and Bowman (1993), Sutherland and Lee (1994) applied the distributions to coastal sediments. In Xu, Durst and Tropea (1993) HYP distributions found application in fluid sprays. Other areas, where HYP distributions have been employed, include biology (e.g. Blæsild (1981)) and primary magnetization of lava flows (Kristjansson and McDougall (1982)). Furthermore, in Barndorff-Nielsen, Jensen and Sørensen (1989) the HYP distribution is employed to model wind shear data of landing aircrafts parsimoniously.

2.3.2 Normal Inverse Gaussian Distributions

The NIG distribution has the following pdf:

$$f_{NIG}(x;\alpha,\beta,\delta,\mu) = \frac{\alpha\delta}{\pi}\frac{K_1\big(\alpha\sqrt{(\delta^2+(x-\mu)^2)}\big)}{\sqrt{\delta^2+(x-\mu)^2}}exp\{\delta\sqrt{\alpha^2-\beta^2}+\beta(x-\mu)\} \tag{2.25}$$

Following an indication in Barndorff-Nielsen (1998), Barndorff-Nielsen, Blæsild and Schmiegel (2004) demonstrated that the NIG distribution is capable of describing velocity data from turbulence experiments with high accuracy. Eriksson, Forsberg and Ghysels (2004) employed the NIG distribution to approximate other unknown probability distributions.

In recent years many authors have successfully fitted NIG distribution to returns in financial time series; see Eberlein and Keller (1995), Prause (1997), Barndorff-Nielsen (1997), Prause (1999), Barndorff-Nielsen and Shephard (2001). This has, in particular, led to modeling the time dynamics of financial markets by stochastic processes using NIG distributions as building blocks.

Figure 2.10 illustrates the comparison between pdf curves of GH distribution and its two subclasses introduced above. The three distributions have mean 0 and variance 1. NIG distribution (-0.5,1,0,1,0) is identified in the graphic as black curve, GH (0.5,1.615,0,1,0) the blue curve and HYP (1,1.875,0,1,0) the red dotted curve, with parameter order $(\lambda,\alpha,\beta,\delta,\mu)$.

2.3.3 Limiting Distributions

An important aspect is that GH distributions cover many special cases, including limiting distributions of normal, Student-t and Cauchy distribu-

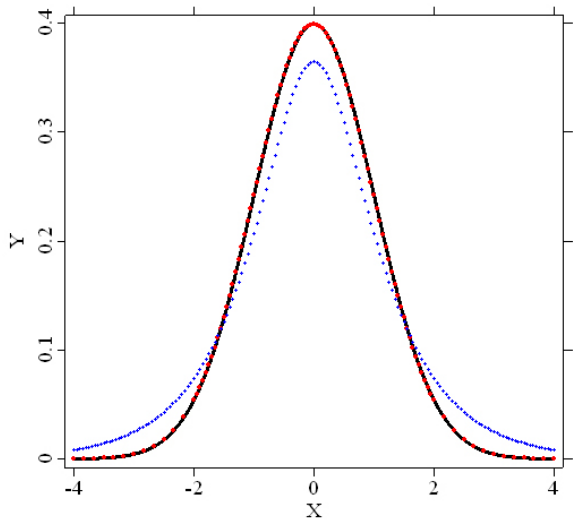

Fig. 2.11: Limiting distribution: normal distributio `lmtGHN.xpl`

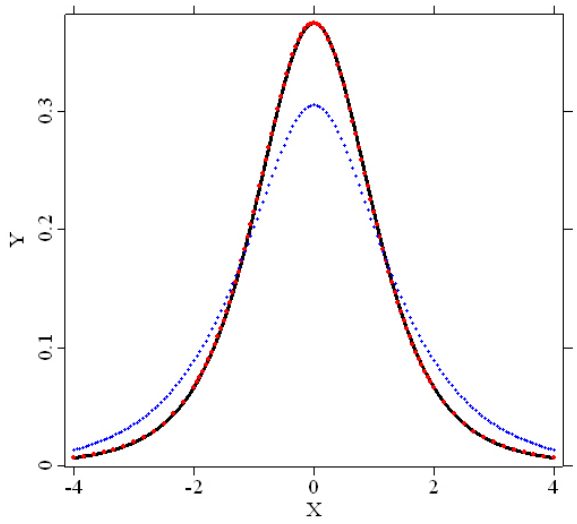

Fig. 2.12: Limiting distribution: t distribution. `lmtGHt.xpl`

tions.

The normal distributions are obtained as a limiting case of the GH distrib-

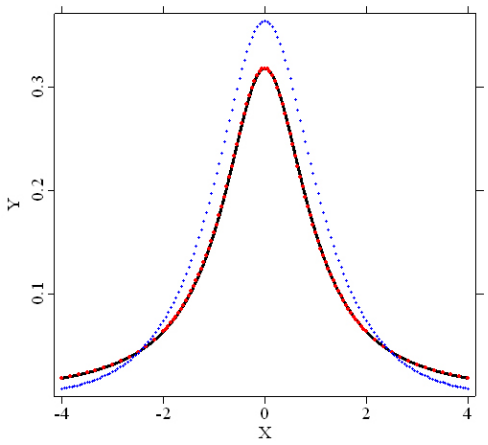

Fig. 2.13: Limiting distribution: Cauchy distributio `lmtGHC.xpl`

utions for $\delta \to \infty$ and $\delta/\alpha \to \sigma^2$. See Barndorff-Nielsen (1978). In Figure 2.11, we set the parameters of GH distribution to (0.5,26,0,26,0), where $\alpha = \delta = 26$, a relatively large value to approximate infinity. The pdf curve of GH(0.5,26,0,26,0) (red dots) laps over the pdf curve of standard normal distribution (black). The blue curve of GH(0.5,1,0,1,0) is placed here as a comparison.

The Student-t distribution results from a mixture of normal and inverse gamma distributions. We have a Student-t distributions as a limit of GH distributions for $\lambda < 0$ and $\alpha = \beta = \mu = 0$. See Barndorff-Nielsen (1978). In Figure 2.12, red dots of GH$(-2, 1.0 \times 10^{-10}, 0, 2, 0)$ lap over the curve of Student-t distribution with degrees of freedom 4, since $\lambda = -n/2$, $\delta = \sqrt{n}$, denoting the degrees of freedom by n. α here is set to a small number to approximate 0. The blue curve stands for GH(1,1,0,1,0) in the graphic.

Cauchy distribution can be obtained from limiting case of GH distributions with $\lambda = -1/2$, $\alpha = \beta = 0$ and $\delta = 1$. See Blæsild (1999). The pdf curve of GH$(-0.5, 1.0 \times 10^{-10}, 0, 1, 0)$ is presented as the red dots in Figure 2.13. The black curve is the pdf curve of standard Cauchy distribution, which is lapped over by the red dots. The blue curve stands for GH(0.5,1,0,1,0) in the graphic.

As a summary, the limiting distributions discussed here are listed in Table 2.1.

	Parameter Description
Normal(μ,δ^2)	$\delta \to \infty$ and $\delta/\alpha \to \sigma^2$.
Student-t(n)	$\lambda < 0$ and $\alpha = \beta = \mu = 0$. $\lambda = -n/2$, $\delta = \sqrt{n}$.
Cauchy(0,1)	$\lambda = -1/2$, $\alpha = \beta = 0$ and $\delta = 1$.

Tab. 2.1: Limiting cases of GH distributions

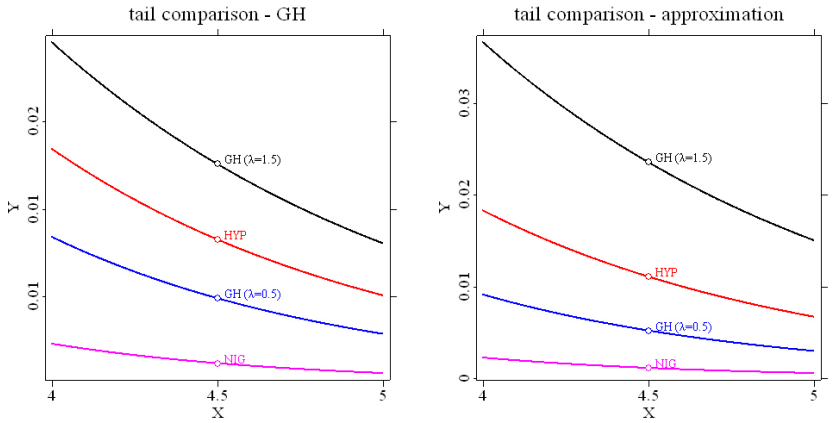

Fig. 2.14: Tail comparison between GH distributions, pdf (left) and approximation (right).

MVAght.xpl

2.4 Tail-Behavior

In the final analysis, it is the heavy tail that makes GH distributions so popular in modelling the time dynamics of financial markets, since it is much closer to the empirical density of financial time series. Generally the GH distributions have an exponential decaying speed

$$f_{GH}(x; \lambda, \alpha, \beta, \delta, \mu = 0) \sim x^{\lambda-1} exp\{-(\alpha - \beta)x\} \qquad \text{as} \quad x \to \infty. \tag{2.26}$$

Figure 2.14 illustrates the tail behavior of GH distributions with different value of λ with $\alpha = 1, \beta = 0, \delta = 1, \mu = 0$. The left panel contains part of pdf curves of GH distributions and the right panel demonstrates the approximation by the function mentioned above. It is clear that among the four distributions, GH with $\lambda = 1.5$ has the lowest decaying speed, while NIG decays fastest.

In Figure 2.15 the tail behavior of GH distributions and the limiting distributions is demonstrated. All distributions have mean 0 and variance 1 except for Student-t, since its variance $\sigma^2 = n/(n-2)$, where n is denoted as the degrees of freedom, and when $n \to \infty$, Student-t approaches normal distribution. The variance of t distribution in the graphic is 1.11 (from $n = 20$).

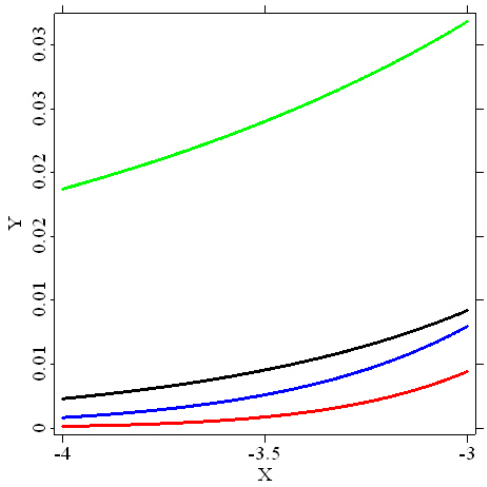

Fig. 2.15: Tail comparison between GH(-0.5,1,0,1,0) distribution (black) and its limiting distributions: normal(red), Student-t (blue) and Cauchy (green) distributions. `lmttail.xpl`

The green curve represents tail of Cauchy distribution, black that of GH distribution, blue Student-t and red normal distribution. Cauchy, GH and Student-t distributions decay more slowly than normal distribution.

3. METHODS OF ESTIMATION

3.1 Maximum-Likelihood Estimation

Assuming the independence of observations x_i, $i = 1, ..., n$, we maximize the log-likelihood function:

$$\begin{aligned} L_{GH}(\lambda,\alpha,\beta,\delta,\mu) &= nlog\{a(\lambda,\alpha,\beta,\delta)\} + (\frac{\lambda}{2} - \frac{1}{4}) \sum_{i=1}^{n} log\{\delta^2 + (x_i - \mu)^2\} \\ &\quad + \sum_{i=1}^{n} \left[logK_{\lambda-\frac{1}{2}}(\alpha\sqrt{\delta^2 + (x_i - \mu)^2}) + \beta(x_i - \mu) \right] \end{aligned} \tag{3.1}$$

For HYP ($\lambda = 1$) or NIG ($\lambda = -1/2$) distributions the algorithm uses the simpler expressions of the log-likelihood function.

Taking the first derivatives of the log-likelihood function respect to the five parameters, we obtain the following expressions, in which the log-likelihood function is denoted by L. See Prause (1999).

$$\begin{aligned} \frac{d}{d\lambda}L &= n\left\{ \frac{1}{2} ln\left(\frac{\alpha^2 - \beta^2}{\alpha\delta} \right) - \frac{k_\lambda(\delta\sqrt{\alpha^2 - \beta^2})}{K_\lambda(\delta\sqrt{\alpha^2 - \beta^2})} \right\} \\ &\quad + \sum_{i=1}^{n} \left\{ \frac{1}{2} ln\{\delta^2 + (x_i - \mu)^2\} + \frac{k_{\lambda-1/2}(\alpha\sqrt{\delta^2 + (x_i - \mu)^2})}{K_{\lambda-1/2}(\alpha\sqrt{\delta^2 + (x_i - \mu)^2})} \right\} \end{aligned} \tag{3.2}$$

$$\begin{aligned} \frac{d}{d\alpha}L &= n\frac{\delta\alpha}{\sqrt{\alpha^2 - \beta^2}} R_\lambda(\delta\sqrt{\alpha^2 - \beta^2}) \\ &\quad - \sum_{i=1}^{n} \sqrt{\delta^2 + (x_i - \mu)^2} R_{\lambda-1/2}(\alpha\sqrt{\delta^2 + (x_i - \mu)^2}) \end{aligned} \tag{3.3}$$

$$\frac{d}{d\beta}L = n\left\{ -\frac{\delta\beta}{\sqrt{\alpha^2-\beta^2}}R_\lambda(\delta\sqrt{\alpha^2-\beta^2}) - \mu \right\} + \sum_{i=1}^{n} x_i \tag{3.4}$$

$$\begin{aligned}\frac{d}{d\delta}L &= n\left\{ -\frac{2\lambda}{\delta} + \sqrt{\alpha^2-\beta^2}R_\lambda(\delta\sqrt{\alpha^2-\beta^2}) \right\} \\ &\quad + \sum_{i=1}^{n}\left\{ \frac{(2\lambda-1)\delta}{\delta^2+(x_i-\mu)^2} - \frac{\alpha\delta R_\lambda(\alpha\sqrt{\delta^2+(x_i-\mu)^2})}{\sqrt{\delta^2+(x_i-\mu)^2}} \right\}\end{aligned} \tag{3.5}$$

$$\begin{aligned}\frac{d}{d\mu}L &= -n\beta + \sum_{i=1}^{n}\frac{x_i-\mu}{\sqrt{\delta^2+(x_i-\mu)^2}} \\ &\quad \times\left\{ \frac{2\lambda-1}{\sqrt{\delta^2+(x_i-\mu)^2}} - \alpha R_{\lambda-1/2}(\alpha\sqrt{\delta^2+(x_i-\mu)^2}) \right\}\end{aligned} \tag{3.6}$$

where

$$\begin{aligned}k_\lambda(x) &= \frac{dK_\lambda(x)}{d\lambda} \\ R_\lambda(x) &= \frac{K_{\lambda+1}(x)}{K_\lambda(x)}\end{aligned}$$

Set them to zero, we obtain a complicated nonlinear equation system. Theoretically, there is a solution to a system with five equations and five unknown parameters. However, in practice, the solution is very difficult to be acquired.

Algorithms without using derivatives are therefore utilized to solve the problem of maximizing a function in five-dimensional space.

3.2 Numerical Algorithms

The algorithms of multi-dimensional maximization require algorithms of one-dimensional search for maximum value. The following algorithms are presented by Press, Teukolsky, Vetterling and Flannery (2002).

3.2.1 Golden Section Search in One Dimension

The golden section or golden mean has its root back to the ancient Pythagoreans. The fraction of 0.38197 or 0.61803 is considered to have some aesthetic properties and it is rather helpful to search for the extremes.

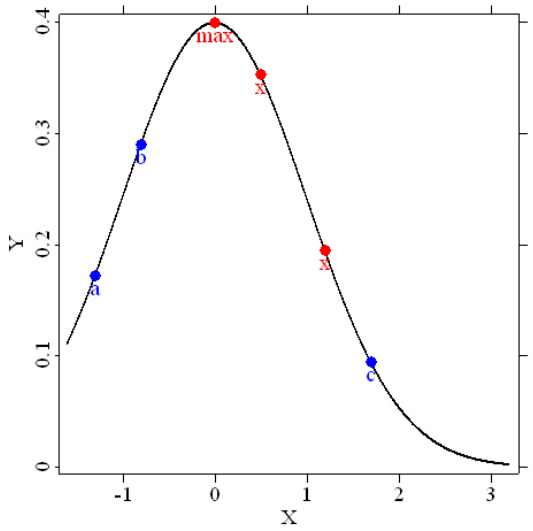

Fig. 3.1: Illustration of successive bracketing of a maximum by golden section search in one dimension. GHgs.xpl

In Figure 3.1, we have a triplet of points: $\{a, f(a)\}, \{b, f(b)\}$ and $\{c, f(c)\}$ with $(a < b < c)$. Together they bracket a maximum. Among the ordinates, $f(b)$ is then the largest one. Now we choose a new point x, either between a and b or b and c. Suppose we choose the latter. Then evaluating of $f(x)$ is crucial for the choice of the next brackting points. If $f(x) < f(b)$, then the new bracketing triplet of points is (a, b, x); if $f(x) > f(b)$, then the new bracketing triplet is (b, x, c). The principle is: the middle point of the new triplet must be the abscissa whose ordinate is the best maximum achieved so far. We continue the process of bracketing until the distance between the two outer points of the triplet is tolerably small. The optimal bracketing interval (a, b, c) has its middle point b a fractional distance 0.38197 from one end, and 0.61803 from the other end. This optimal method of function minimization is thus called *golden section search.*

Golden section search can be summarized as follows:

At each stage, given a bracketing triplet of points, the next point to be tried is that which is a fraction 0.38197 into the larger of the two intervals (measuring from the center point of the triplet). Even we are in a situation that the segments of starting bracketing triplet are not in the golden ratios, the procedure of choosing successive points at the golden mean point of the larger segment will quickly converge us to the proper ratios. By using golden section search, it is guaranteed that we will bracket the maximum to

an interval just 0.61803 times the size of the preceding interval.

Press et al. (2002) demonstrate why the golden section is the optimal fraction when searching for the maximum. Suppose that b is a fraction w of the way between a and c, i.e.

$$\frac{b-a}{c-a} = w \qquad \frac{c-b}{c-a} = 1-w. \tag{3.7}$$

Also suppose that our next trial point x is an additional fraction beyond b,

$$\frac{x-b}{c-a} = z. \tag{3.8}$$

Then the length of the next bracketing segment will either be $w+z$ or $1-w$, relative to the current one. In order to minimize the worst case possibility, we choose z to make these equal, namely

$$z = 1-2w. \tag{3.9}$$

It is easy to find out that the equation 3.9 implies that the point x lies in the larger of the two segments, since z is positive only if $w < 1/2$. And the new point is symmetric point to b in the original interval, namely with $|b-a|$ equal to $|x-c|$.

Suppose we apply the same strategy at each stage, then if z is chosen to be optimal, so was w before it. This scale similarity implies that x should be the same fraction of the way from b to c, if that is the larger segment, as was b from a to c, in other words,

$$\frac{z}{1-w} = w. \tag{3.10}$$

Equation 3.9 and 3.10 give the quadratic equation

$$w^2 - 3w + 1 = 0, \tag{3.11}$$

which yields:

$$w = \frac{3-\sqrt{5}}{2} \approx 0.38197 \tag{3.12}$$

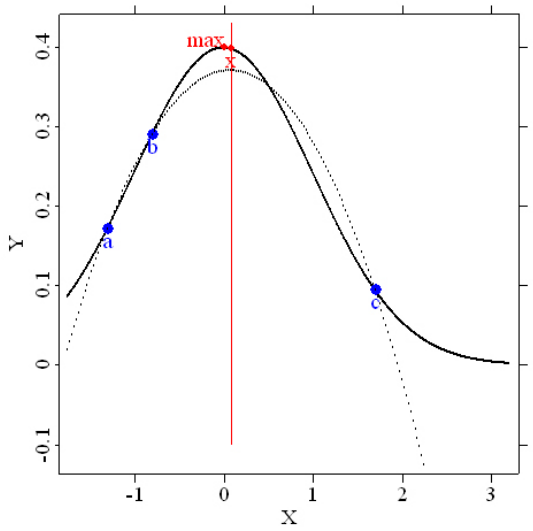

Fig. 3.2: Illustration of successive bracketing of inverse parabolic interpolation in one dimension. GHpi.xpl

3.2.2 Parabolic Interpolation and Brent's Method in One Dimension

Another one-dimensional algorithm presented by Press et al. (2002) to search for a extreme is parabolic interpolation. Before we go on to this algorithm, let's take a closer look at golden section search.

The introduction to golden section search and the deduction of golden mean makes it clear that golden section search is designed to deal with the worst case of function maximization. Although it is a sure process, it is very slow. If the function is nicely parabolic near to the maximum, then we do not need to slowly crawl through the whole function, the parabola fitted through any three points ought to take us in a single leap to the maximum, or at least very near to it (see Figure 3.2). The procedure is technically called inverse parabolic interpolation, since we want to find an abscissa rather than an ordinate.

The formula for the abscissa x that is the maximum of a parabola through three points $\{a, f(a)\}, \{b, f(b)\}$ and $\{c, f(c)\}$ is:

$$x = b - \frac{1}{2}\frac{(b-a)^2\{f(b)-f(c)\} - (b-c)^2\{f(b)-f(a)\}}{(b-a)\{f(b)-f(c)\} - (b-c)\{f(b)-f(a)\}}. \tag{3.13}$$

Since this formula involves a fraction, it should be noticed that the fomula

fails if the denominator is zero, which means the three points are collinear.

However no maximization scheme that depends solely on 3.13 is likely to succeed. We should combine the strengths of two approaches in order to succeed. The exacting task is to invent a scheme that relies on a sure-but-slow technique, like golden section search mentioned above, when the function is not cooperative, but that switches over to a quick search, like parabolic interpolation introduced in this section, when the function allows. Brent's method is implemented as the scheme. The general principles are: the parabolic step must fall within the bounding interval (a, b), and imply a movement from the best current value x that is less than half the movement of the step before last. This second criterion insures that the parabolic steps are actually converging to something, rather than, for example, bouncing around in some nonconvergent limit cycle. Press et al. (2002) give an empirical reason for comparing to the step before last: Experience shows that it is better not to "punish" the algorithm for a single bad step if it can make it up on the next one.

3.2.3 Powell's Methods in Multidimensions

The fundament of multidimensional search for extreme is still the algorithms of one-dimension. Suppose we start at a point $\mathbf{P}$ in N-dimensional space, and proceed from there in some vector direction $\mathbf{n}$, then any function of N variables $f(\mathbf{P})$ can be maximized along the line $\mathbf{n}$ by our one-dimensional methods. When the maximum is achieved along $\mathbf{n}$, a new direction is chosen, and the maximizing process is repeated along the new direction. The multidimensional maximization therefore consists of successive sequences of such line maximization. Different methods will differ only by how, at each stage, they choose the next direction $\mathbf{n}$ to try.

As demonstrated by Press et al. (2002) in their book, a good direction searching method requires that maximization along one direction is not undermined by the subsequent maximization along another. This kind of directions is called "non-interfering" direction, or more conventionally conjugate directions.

Suppose that we are going to move along some new direction $\mathbf{v}$ after having moved along some direction $\mathbf{u}$ to a maximum. The condition that motion along $\mathbf{v}$ not spoil our maximization along $\mathbf{u}$ is just that the gradient stay perpendicular to $\mathbf{u}$. Two vectors $\mathbf{u}$ and $\mathbf{v}$ are said to be conjugate, when the relation mentioned above holds for them. And a set of vectors is called a conjugate set, when the relation holds pairwise for all members of the set of vectors. If we do successive line maximization of a function along a conjugate set of directions, then we don't need to redo any of those directions.

Powell first discovered a direction set method that does produce N mutually conjugate directions. The basic procedure is given as follows:

First, initialize the set of direction $\mathbf{u}_i$ to the basis vectors,

$$\mathbf{u}_i = \mathbf{e}_i \qquad i = 0, ..., N-1. \tag{3.14}$$

Now repeat the following sequence of steps until the function stops decreasing:

- Save the starting position as $\mathbf{P}_0$
- For $i = 0, ..., N-1$, move $\mathbf{P}_i$ to the minimum along direction $\mathbf{u}_i$ and call this point $\mathbf{P}_{i+1}$.
- For $i = 0, ..., N-2$, set $\mathbf{u}_i \leftarrow \mathbf{u}_{i+1}$.
- Set $\mathbf{u}_{N-1} \leftarrow \mathbf{P}_N - \mathbf{P}_0$.
- Move $\mathbf{P}_N$ to the minimum along direction $\mathbf{u}_{N-1}$ and call this point $\mathbf{P}_0$.

The basic procedure is given to search for a minimum, an analog of search for maximum.

Powell, in 1964, showed that, for a quadratic form like Taylor series, k iterations of the above basic procedure produce a set of directions $\mathbf{u}_i$ whose last k members are mutually conjugate. Therefore, N iterations of basic procedure, amounting to $N(N+1)$ line maximizations in all, will exactly minimize a quadratic form.

When we implement Powell's quadratically convergent algorithm, a problem emerges. The procedure of throwing away, at each stage, $\mathbf{u}_0$) in favor of $\mathbf{P}_N - \mathbf{P}_0$ tend to produce sets of directions that fold up on each other and become linearly dependent. Once this happens, then the procedure finds the minimum of the function f only over a subspace of the full N-dimensional case.

In order to fix up the problem of linear dependence in Powell's algorithm, we implement a method which tries to tind a few good directions instead of N necessary conjugate directions.

The basic idea of our modified Powell's method is still to take $\mathbf{P}_N - \mathbf{P}_0$ as a new direction; it is, after all, the average direction moved after trying all N possible directions. The change is to discard the old direction along which the function f made its largest decrease. This seems paradoxical, since that

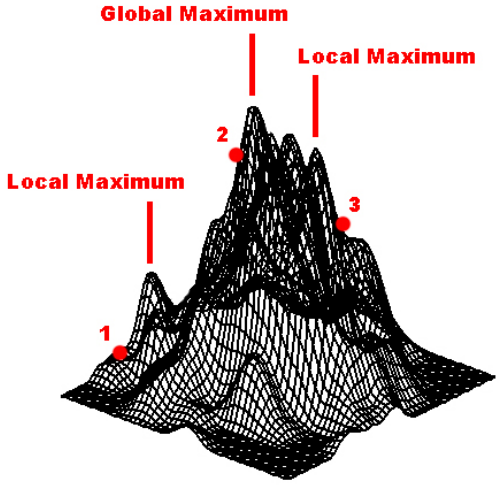

Fig. 3.3: Local and global maximum.

direction was the best of the previous iteration. However, it is also likely to be a major component of the new direction that we are adding, so dropping it gives us the best chance of avoiding a buildup of linear dependence.

3.3 Local and Global Maximum

When maximizing a function, the problem of local and global maximum arises. If, for instance, a two-dimensional function forms a surface as illustrated in Figure 3.3. Besides the global maximum in the middle, there are many local maximums (identified as many small peaks in the graphic). Suppose we start our process from the point 2, and then probably we will reach the global maximum. It is the optimal case. If we start from point 3, we will probably achieve a local maximum near the global one. But that is not the worst case; the difference between the value of global maximum and local maximum marked in the graphic besides point 3 is not so large. The fall between the global maximum and the local maximum besides point 1 is quite large. If this is the surface of a likelihood function and we start from point 1, then we will get estimated parameters far from genuine ones.

Without knowledge of the structure of the probability density, it is rather difficult to optimize the choice of starting position. A partial solution will be choosing different starting points and comparing the resulting maximums. In the following chapter we will discuss on the robustness of starting position.

4. SIMULATION STUDIES

4.1 Procedure

We now utilize simulations to assess our results of estimation. The procedure is described as follows:

- Generate a set of artificial observations which is GH distributed. See rndgh.xpl.
- Estimate the parameters by the 5-parameter estimation methods: mlgh.xpl or mlghint.xpl. See 7.2.
- Repeat the first two steps for many times to obtain a batch of estimated parameters.
- Compare the estimated parameters with the original ones.

The details of implementation of estimation can be found in Appendix 7.1.

Here, to be specific, we generate 2000 artificial observations and repeat the first two steps for 200 times to obtain a batch of estimated parameter with dimension 200×5.

Table 4.1 shows 15 results of estimation with the original parameters (-1,1,0,1,0) (We use the parameter order $(\lambda, \alpha, \beta, \delta, \mu)$). The last column of the table records the log-likelihood value.

4.2 Estimation with Fractional λ

We start our assessment of estimation with original parameters (-1,1,0,1,0). 200 repetitions provide us with results demonstrated in Table 4.1. The first row is the original parameters used to generate artificial data. The row labeled "mean" records mean values of all 200 sets of estimated parameters. The last row gives standard deviations of the estimated parameters.

Table 4.1 indicates, that the means of the estimated parameters are very close to the original parameters. Figure 4.3 corroborates the conclusion.

	λ	α	β	δ	μ	LogLH
original	-1	1	0	1	0	
01	-1.005116	1.132264	0.099794	1.081788	-0.051375	-2413.710677
02	-1.045258	0.987556	0.093914	0.982782	-0.046093	-2342.614970
03	-0.919434	0.857116	0.000584	0.908057	-0.008591	-2405.195723
04	-1.010331	1.026054	0.010411	1.033796	-0.028264	-2406.453442
05	-0.955805	0.976756	-0.040283	0.989247	0.005690	-2416.353935
06	-0.150855	1.442824	-0.075099	0.785681	0.030091	-2381.657109
07	-1.000162	0.958336	-0.016290	0.989951	0.029794	-2395.811771
08	-1.109315	1.217199	-0.050687	1.109509	0.055419	-2338.019558
09	-1.332997	0.627534	-0.088322	1.019972	0.018861	-2391.760526
10	-1.078312	1.039969	-0.040200	1.023171	0.002410	-2341.156827
11	-0.439145	1.222142	0.108989	0.804611	-0.056101	-2373.758945
12	-1.082249	1.219899	0.045270	1.114007	-0.029067	-2356.557833
13	-0.956055	1.000209	-0.072299	0.992588	0.051187	-2408.668955
14	-1.203670	0.968539	-0.039005	1.054661	0.016276	-2342.677160
15	-0.701177	1.164781	-0.068984	0.935313	0.026270	-2398.894757
...	...	...	...	...	...	...
mean	-0.989951	1.019463	0.001551	1.003807	-0.003007	-2377.004453
s.d.	0.534790	0.311415	0.057220	0.155158	0.033739	43.654212

Tab. 4.1: Example: Results of Estimation

The left panel of the graphic compares the original (blue) with estimated (blue) pdf curves, while the right panel concentrates on the left tail area of the distributions. In either panel, the red and blue curves overlap, indicating that the results of estimation are truly satisfactory.

However, Figure 4.1 shows that there are many outliers for estimated λ and α. A closer look at Table 4.1 reveals more information. Among the 15 examples of estimated parameters, 6th and 11th are striking. The estimated parameters deviate relatively far from the mean. From the values of the table, it seems that their pdf curves have little chance to be adjacent to the original one, but the flexibility of GH distributions allows the effects of one parameter to be compensated by other parameters. The results are illustrated in Figure 4.4 and Figure 4.5. The graphics show that their pdf curves almost overlap with the original ones. Their tails seem to be far from the original ones in the graphics, but the scale of y-axis clarifies that they are actually very close to the original tails. The results are helpful to explain the standard deviations in Table 4.1 and the many outliers of λ and α in Figure 4.1.

Now we gradually increase the value of λ and measure the results of estimation. Table 4.2 lists the results when λ takes four different values, and

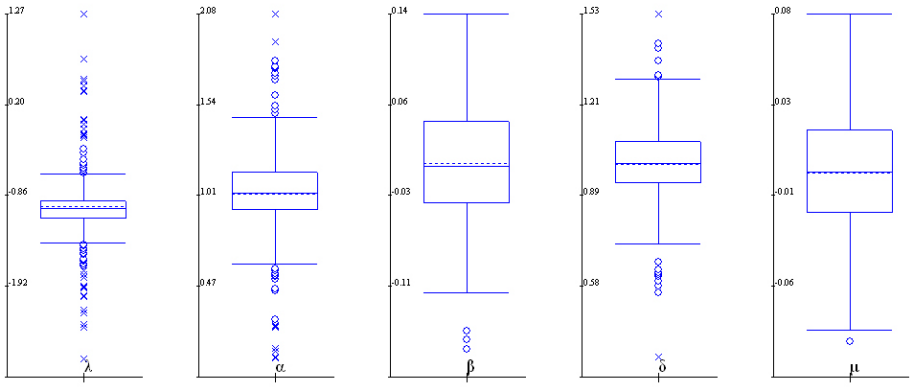

Fig. 4.1: Boxplot of Example (-1,1,0,1,0) GHbox.xpl

Figure 4.6 - 4.9 compare their estimated pdf curves with the original ones. When λ takes the value of -0.5 or 1, GH distribution is identified as its two subclasses. See Chapter 2.3.

Either the table or the figures indicate, that the results of estimation are desirable, when original λ takes the value of -3, -1, -0.5, 0.49 and 1. When original $\lambda = -3$, all parameters fit quite well. The largest deviation comes from λ. There is less than 0.11 difference between the original and estimated λ. Figure 4.2 demonstrates two overlapped curves, implying a perfect match, and the right panel shows that the difference between two tails is measured by the magnitude of 10^{-4}. When original $\lambda = -0.5$ or $\lambda = 0.49$, either the values of estimated parameters or the pdf curves are very close to the original ones. In these cases, there is nothing to carp at. For the original $\lambda = 1$, estimated λ deviates a little from 1, while other parameters are still very close to the original ones. As illustrated by Figure 4.8, a small gap appears between the red and blue curves, indicating a deficient value of estimated λ.

When original λ takes a large positive value, in our case, $\lambda = 3$, the estimated and original pdf curves separate, the estimated pdf curve is more peaked in the middle than the original curve. The estimated value of λ is much smaller. It has a value around 1.03, only a third of the original one. Although the value of δ increases and takes a value more than 2.2, it is not large enough to offset the effects of λ. A change of estimation order of parameters does not improve the results.

The influences of α on the results of estimation can also be evaluated in the same way. Here we choose original $\lambda = -0.5$ and $\lambda = 1$, so that our results

are more comparable with other literature. We increase the value of α to 2 and then to a large one, 4.5. Table 4.3 - 4.4 and Figure 4.10 - 4.13 illustrate the results.

Table 4.3 shows rather large deviation of λ and α, while other parameters are well estimated. When original $\alpha = 2$, estimated α is a little smaller than the original one, and estimated λ take a much smaller negative value than -0.5. But the results are satisfactory. As displayed in Figure 4.10, most parts of the curves overlaps, and the only gap at the peak is also quite small. The two tails are very close to each other. The effects of deviations offset each other, providing us a nice result. When original $\alpha = 4.5$, estimated λ goes even deeper into a small negative value, far from the original, while estimated α takes a value only more than a half of 4.5. The combined results are illustrated in Figure 4.11. The gap at the peak is larger than that with $\alpha = 2$ and the vertical difference between tails is larger than 0.005.

Table 4.4 displays a similar way of deviation as Table 4.3. The performance of estimation is worse in the case of $\lambda = 1$ than in the case of $\lambda = -0.5$, as illustrated by Figure 4.12 and Figure 4.13. However the differences of tails are relatively small. They are measured with a magnitude of 10^{-3}.

In Table 4.5, we use a positive β to examine the performance of estimation of a skewed GH distribution. 200 repetitions provide us with perfect results of estimation when original $\lambda = -0.5$. The estimated parameters are very close to the original ones. Figure 4.14 verifies this with two overlapped curves. The results are not so perfect when $\lambda = 1$, as Figure 4.15 indicates. There are gaps between the estimated and original curves, although the left tails are still very close to each other.

From the simulation studies, it can be concluded that the estimation of GH distributions works very well, when the parameters of GH distributions are not very large. However, when the original parameters, particularly λ and α, take large positive values, the performances of estimation are not so desirable.

	λ	α	β	δ	μ
original	-	1	0	1	0
mean($\lambda = -3$)	-2.894749	1.067013	-0.026335	0.989045	0.002919
s.d.	0.684444	0.713504	0.143434	0.110281	0.031640
mean($\lambda = -0.5$)	-0.480997	1.013325	-0.006658	0.991011	0.000318
s.d.	0.505728	0.217238	0.041726	0.207311	0.036485
mean($\lambda = 0.49$)	0.546592	0.997767	-0.004014	0.943336	-0.000887
s.d.	0.351150	0.096753	0.032694	0.265398	0.058887
mean($\lambda = 1$)	0.678136	0.976144	-0.002571	0.988370	0.001080
s.d.	0.421271	0.104162	0.031640	0.310164	0.064086
mean($\lambda = 3$)	1.032258	1.101515	-0.003794	2.240160	0.001590
s.d.	0.561137	0.121657	0.043518	0.542232	0.145489

Tab. 4.2: Results of estimation with fractional λ, original parameters (λ,1,0,1,0).

	λ	α	β	δ	μ
original	-0.5	-	0	1	0
mean($\alpha = 2$)	-1.330474	1.585631	-0.012431	1.147776	0.001740
s.d.	0.905404	0.429256	0.081769	0.263837	0.039773
mean($\alpha = 4.5$)	-3.878362	2.434933	-0.007591	1.173874	-0.001036
s.d.	0.745779	0.951038	0.257595	0.132516	0.046358

Tab. 4.3: Results of estimation with fractional λ, original parameters (-0.5,α,0,1,0).

	λ	α	β	δ	μ
original	1	-	0	1	0
mean($\alpha = 2$)	-1.171250	1.435458	-0.002609	1.365879	-0.002931
s.d.	0.547099	0.262497	0.080305	0.194302	0.053917
mean($\alpha = 4.5$)	-3.960707	3.009104	-0.001979	1.343088	-0.002386
s.d.	0.660589	0.931344	0.277702	0.169194	0.057585

Tab. 4.4: Results of estimation with fractional λ, original parameters (1,α,0,1,0).

	λ	α	β	δ	μ
original	-	1.5	0.5	1	0
mean($\lambda = -0.5$)	-0.642239	1.455295	0.504557	1.038165	-0.005575
s.d.	0.548510	0.245575	0.083440	0.188165	0.046624
mean($\lambda = 1$)	0.181915	1.377126	0.510125	1.116492	-0.010979
s.d.	0.514431	0.157081	0.080562	0.285200	0.077451

Tab. 4.5: Results of estimation with fractional λ, original parameters (λ,1.5,0.5,1,0).

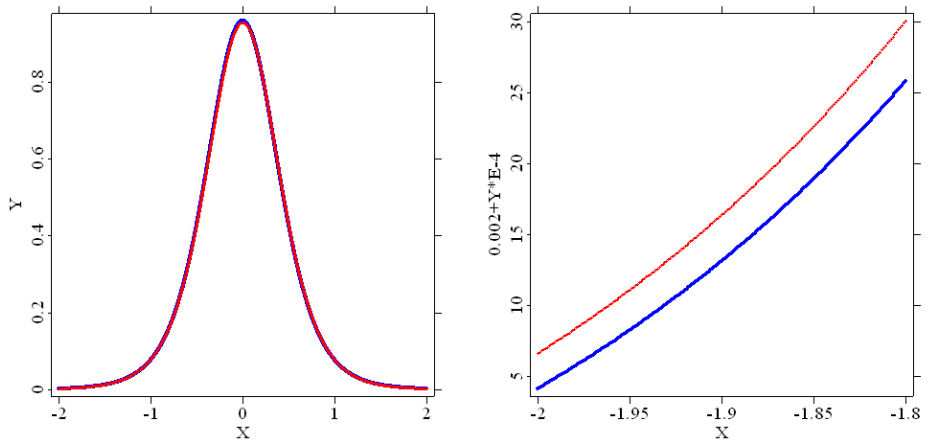

Fig. 4.2: Comparison between original (blue) and estimated (red) pdf curves with parameters (-3,1,0,1,0) and fractional λ.

`simcmp00.xpl`

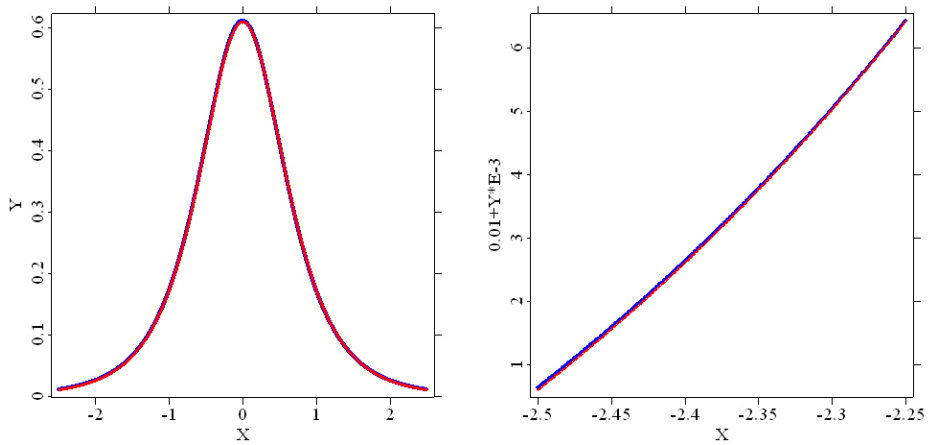

Fig. 4.3: Comparison between original (blue) and estimated (red) pdf curves with parameters (-1,1,0,1,0) and fractional λ.

`simcmp01.xpl`

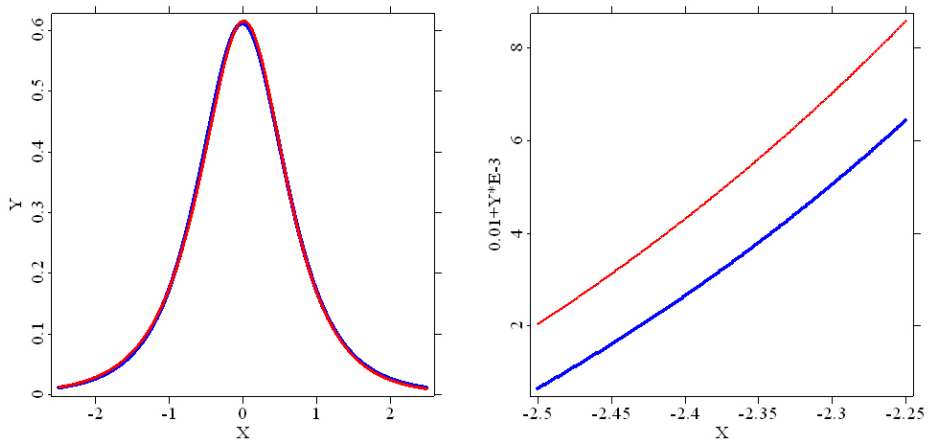

Fig. 4.4: Comparison between original (blue) and 6th estimated (red) pdf curves with parameters (-1,1,0,1,0) and fractional λ. `dev1.xpl`

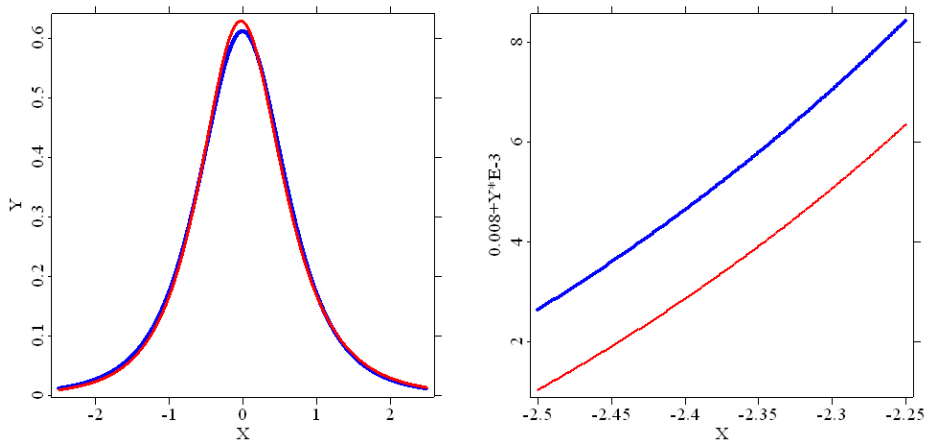

Fig. 4.5: Comparison between original (blue) and 11th estimated (red) pdf curves with parameters (-1,1,0,1,0) and fractional λ. `dev2.xpl`

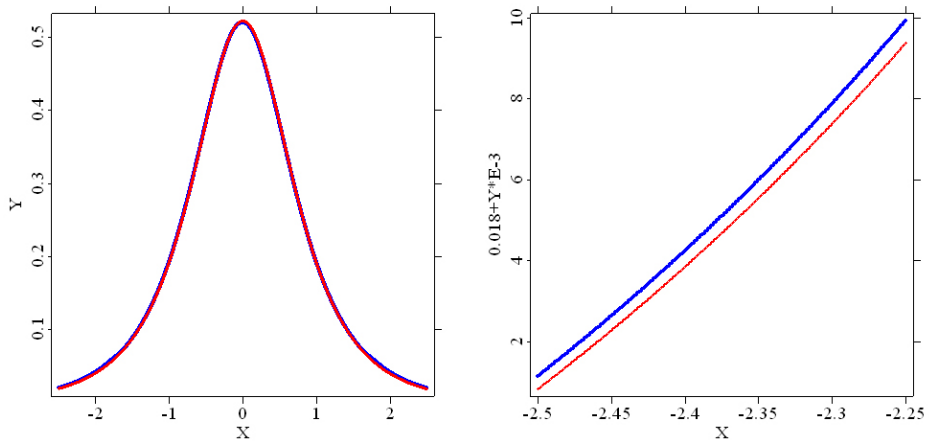

Fig. 4.6: Comparison between original (blue) and estimated (red) pdf curves with parameters (-0.5,1,0,1,0) and fractional λ. simcmp02.xpl

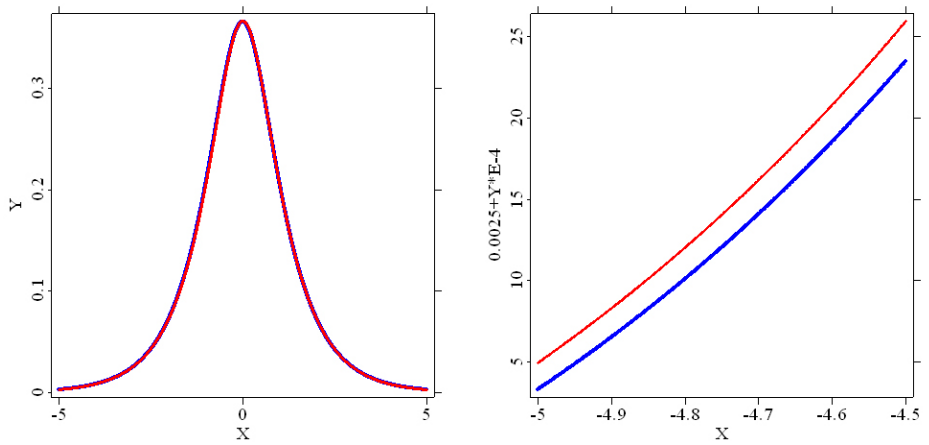

Fig. 4.7: Comparison between original (blue) and estimated (red) pdf curves with parameters (0.49,1,0,1,0) and fractional λ. simcmp03.xpl

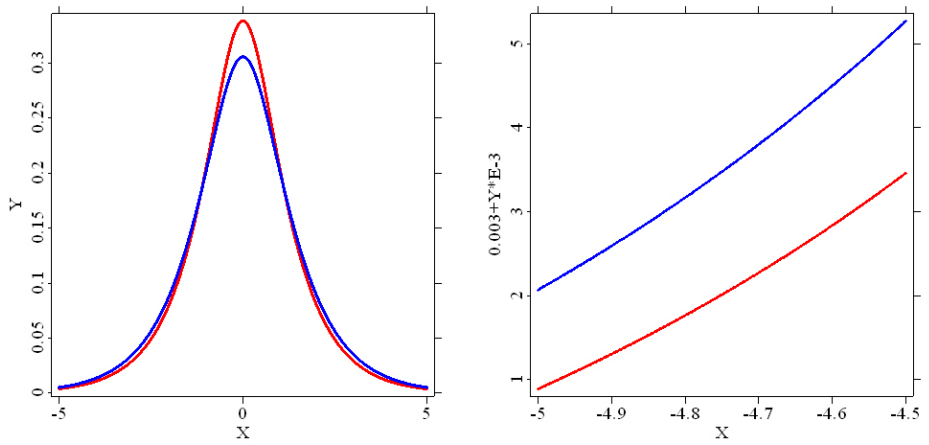

Fig. 4.8: Comparison between original (blue) and estimated (red) pdf curves with parameters (1,1,0,1,0) and fractional λ.

`simcmp04.xpl`

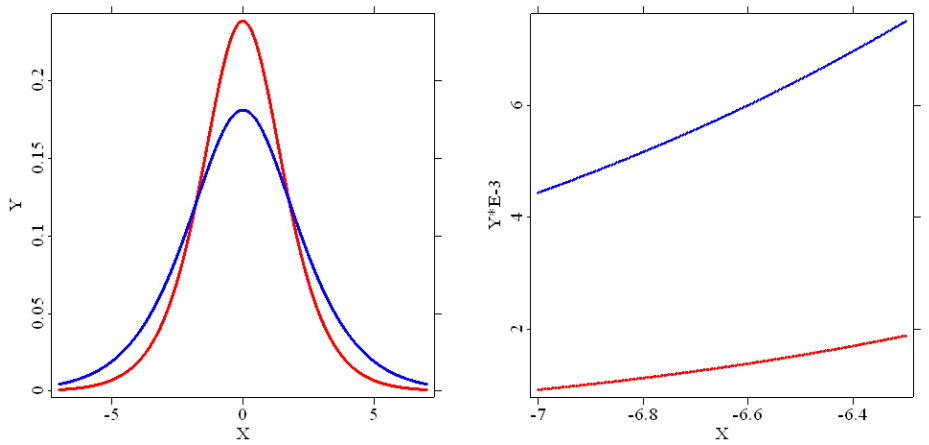

Fig. 4.9: Comparison between original (blue) and estimated (red) pdf curves with parameters (3,1,0,1,0) and fractional λ.

`simcmp05.xpl`

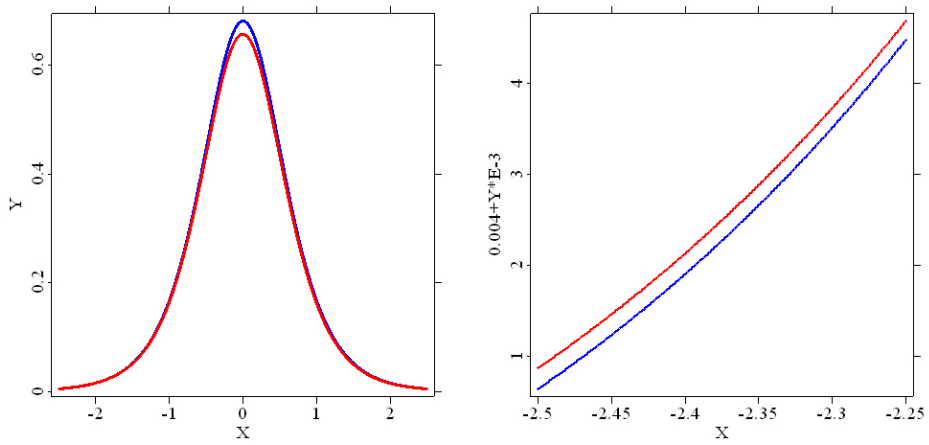

Fig. 4.10: Comparison between original (blue) and estimated (red) pdf curves with parameters (-0.5,2,0,1,0) and fractional λ. `simcmp06.xpl`

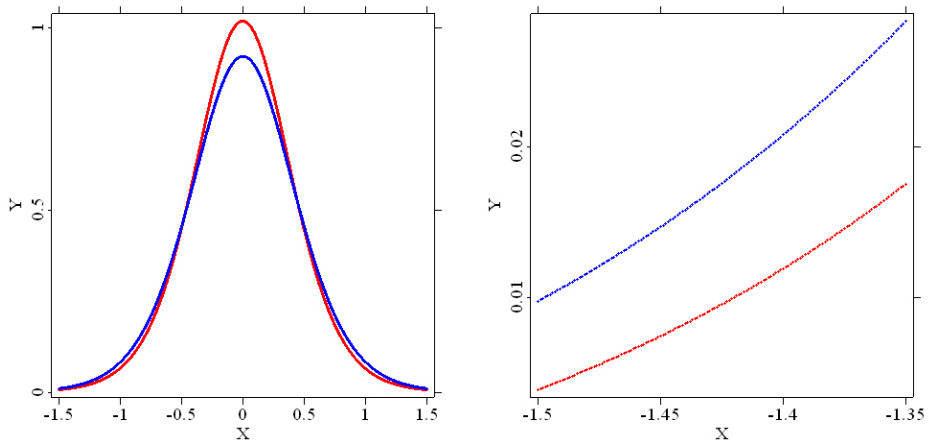

Fig. 4.11: Comparison between original (blue) and estimated (red) pdf curves with parameters (-0.5,4.5,0,1,0) and fractional λ. `simcmp07.xpl`

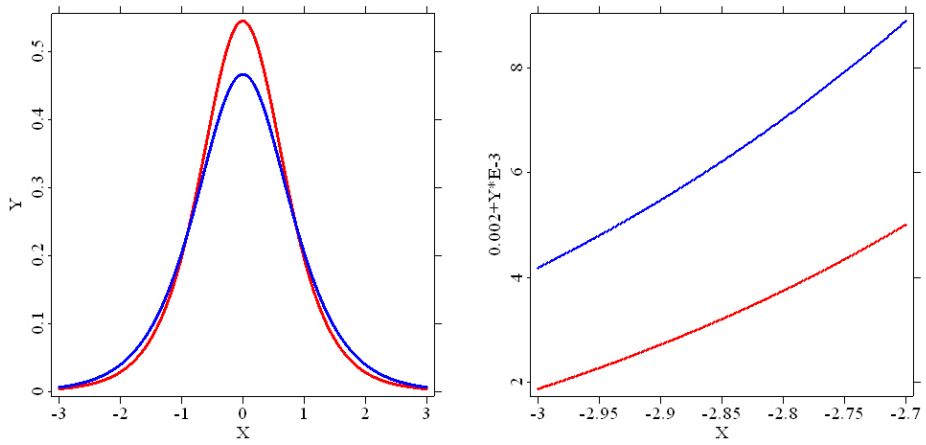

Fig. 4.12: Comparison between original (blue) and estimated (red) pdf curves with parameters (1,2,0,1,0) and fractional λ.

`simcmp08.xpl`

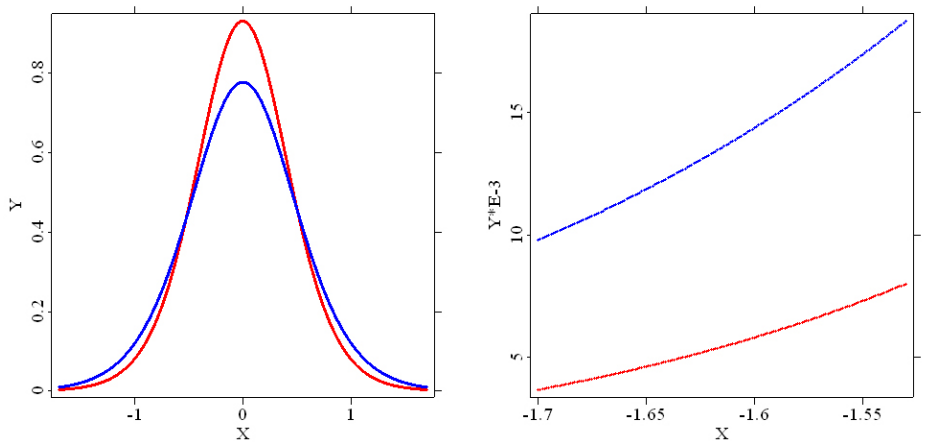

Fig. 4.13: Comparison between original (blue) and estimated (red) pdf curves with parameters (1,4.5,0,1,0) and fractional λ.

`simcmp09.xpl`

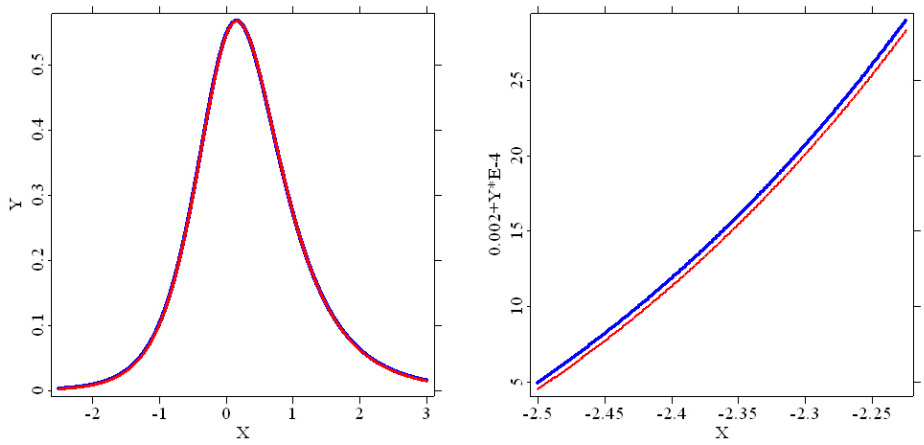

Fig. 4.14: Comparison between original (blue) and estimated (red) pdf curves with parameters (-0.5,1.5,0.5,1,0) and fractional λ. simcmp10.xpl

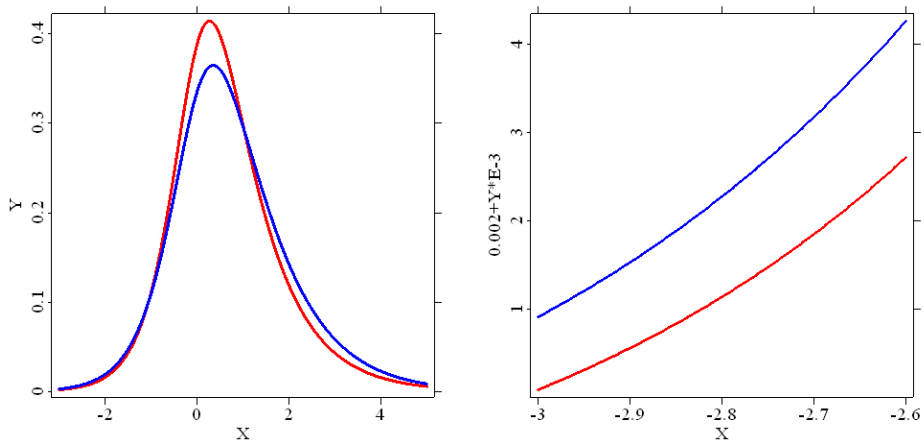

Fig. 4.15: Comparison between original (blue) and estimated (red) pdf curves with parameters (1,1.5,0.5,1,0) and fractional λ. simcmp11.xpl

4.3 Estimation with integer λ

Estimation of GH distributions is time consuming. The main reason is the complexity to compute modified Bessel function of fractional order. If we use a modified Bessel function of integer order, the costs of estimation will be substantially reduced. This means that we use an integer λ to replace the real λ which we used above. The details of implementation can be found in Appendix 7.1.

The assessment of estimation with integer λ uses the same batch of original parameters as in the last section.

Table 4.6 and Figure 4.16 - 4.21 demonstrate the results of estimation with different original λs. One feature of Table 4.6 is really striking: except the case when original $\lambda = 3$ the estimated λs all take the value of 1. In the only exception, the estimated $\lambda = 1.92$, the mean value of 16 λs which take the value of 1 and 184 λs which take the value of 2 among the 200 simulations. On the other hand, estimated αs and δs deviate far from the original values.

However, the total results are quite desirable when original λ takes the value of -3, -1, -0.5, 0.49 and 1, as illustrated in Figure 4.16 - 4.20. In these cases, most parts of the estimated and original pdf curves overlap and their magnified tail parts are also very close to each other. The tiny gaps at the peaks and relatively larger distance between estimated and original tails in the Figure 4.16 - 4.19 indicate that estimation with integer λ performs slightly worse than that of fractional λ. When original $\lambda = 1$, the gap at the peak occurs as before. From the estimated parameters, we can find out that δ is a little smaller than the original value while other parameters are well estimated. The gap is created solely by the deficient δ estimates. When λ takes a large positive value, as in Figure 4.21 $\lambda = 3$, the results are not acceptable as well.

Table 4.7 - 4.9 and Figure 4.22 - 4.27 show the outcome of estimation with different original αs and positive β. The tables show that while λ always takes the value of 1, estimated αs and δs deviate from the original values to make the estimated pdf curves adjacent to the original ones.

For original parameter $\lambda = -0.5$, we get satisfactory results for $\alpha = 2$ and $\alpha = 4.5$, while the performance is not so desirable for the case $\lambda = 1$. When original β takes a positive value, estimated βs are very close to the original one. Figure 4.22 - 4.27 illustrates, that the outcome we obtain about α and β by estimation with integer λ is similar to that with fractional λ.

Generally speaking, the estimation with fractional λ performs better than that with integer λ, but the latter takes much less time to compute.

	λ	α	β	δ	μ
original	-	1	0	1	0
mean($\lambda = -3$)	1.000000	4.019720	-0.002461	0.493896	-0.001056
s.d.	0.000000	0.318593	0.159888	0.086667	0.033514
mean($\lambda = -1$)	1.000000	1.930011	-0.006522	0.418007	-0.000061
s.d.	0.000000	0.090105	0.066943	0.084313	0.038530
mean($\lambda = -0.5$)	1.000000	1.589057	-0.002052	0.452279	-0.001131
s.d.	0.000000	0.072576	0.048818	0.092661	0.039207
mean($\lambda = 0.49$)	1.000000	1.109968	0.000763	0.681548	-0.004715
s.d.	0.000000	0.054870	0.033982	0.154060	0.058843
mean($\lambda = 1$)	1.000000	1.050600	-0.003197	0.812155	0.001792
s.d.	0.000000	0.055246	0.030303	0.178600	0.061626
mean($\lambda = 3$)	1.920000	1.238800	-0.002404	1.797543	0.001924
s.d.	0.271974	0.098074	0.042012	0.549621	0.129113

Tab. 4.6: Results of estimation with integer λ, original parameters (λ,1,0,1,0).

	λ	α	β	δ	μ
original	-0.5	-	0	1	0
mean($\alpha = 2$)	1.000000	2.589086	0.001074	0.665730	-0.002209
s.d.	0.000000	0.186796	0.092427	0.116603	0.042022
mean($\alpha = 4.5$)	1.000000	5.322449	0.015300	0.711570	-0.003989
s.d.	0.000000	0.587208	0.254687	0.137863	0.046108

Tab. 4.7: Results of estimation with integer λ, original parameters (-0.5,α,0,1,0).

	λ	α	β	δ	μ
original	1	-	0	1	0
mean($\alpha = 2$)	1.000000	2.213086	-0.001887	0.840281	-0.001585
s.d.	0.000000	0.152440	0.082170	0.141899	0.054799
mean($\alpha = 4.5$)	1.000000	5.615335	0.003632	0.927466	-0.002235
s.d.	0.000000	0.796738	0.281883	0.199529	0.058265

Tab. 4.8: Results of estimation with integer λ, original parameters (1,α,0,1,0).

	λ	α	β	δ	μ
original	-	1.5	0.5	1	0
mean($\lambda = -0.5$)	1.000000	2.076111	0.527648	0.562929	-0.015993
s.d.	0.000000	0.144388	0.100483	0.106043	0.056133
mean($\lambda = 1$)	1.000000	1.597358	0.521914	0.798836	-0.021656
s.d.	0.000000	0.117911	0.084639	0.141009	0.080893

Tab. 4.9: Results of estimation with integer λ, original parameters (λ,1.5,0.5,1,0).

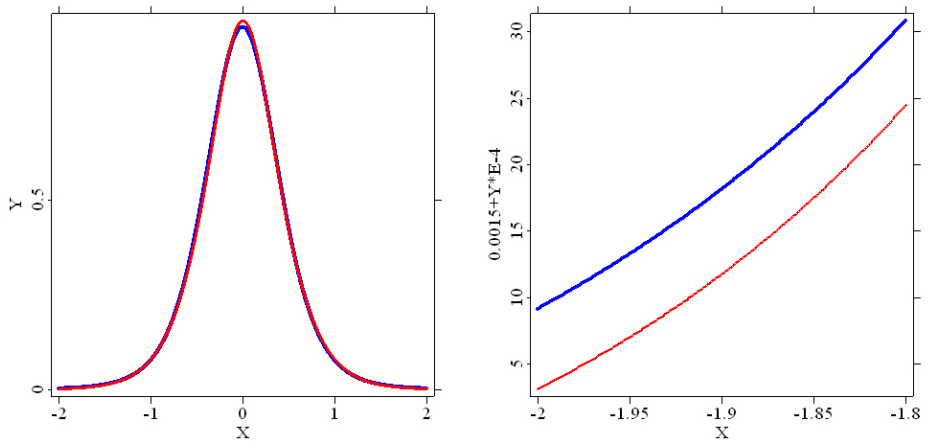

Fig. 4.16: Comparison between original (blue) and estimated (red) pdf curves with parameters (-1,1,0,1,0) and integer λ.
`simint00.xpl`

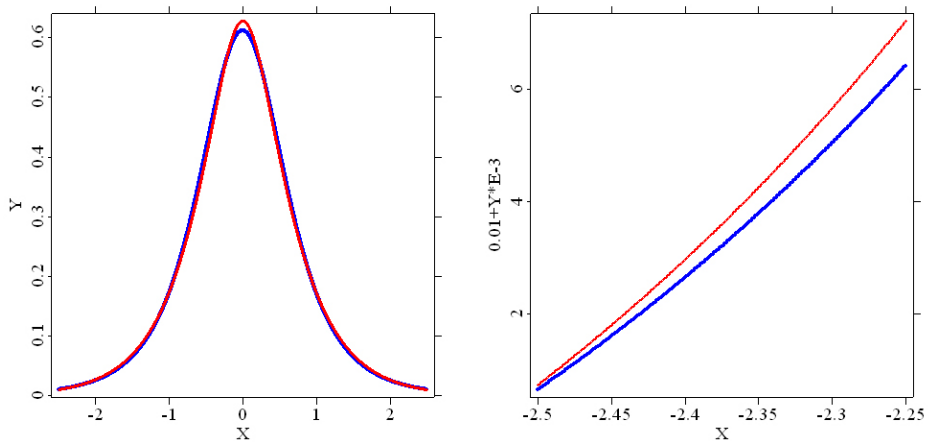

Fig. 4.17: Comparison between original (blue) and estimated (red) pdf curves with parameters (-1,1,0,1,0) and integer λ.
`simint01.xpl`

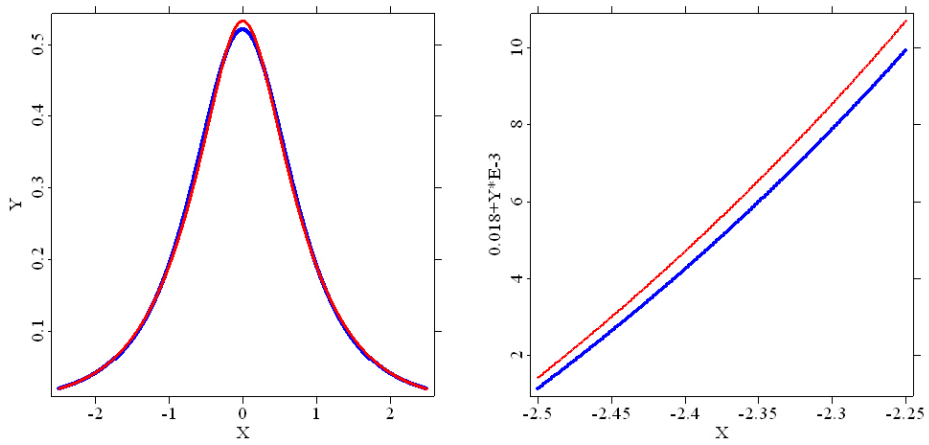

Fig. 4.18: Comparison between original (blue) and estimated (red) pdf curves with parameters (-0.5,1,0,1,0) and integer λ. `simint02.xpl`

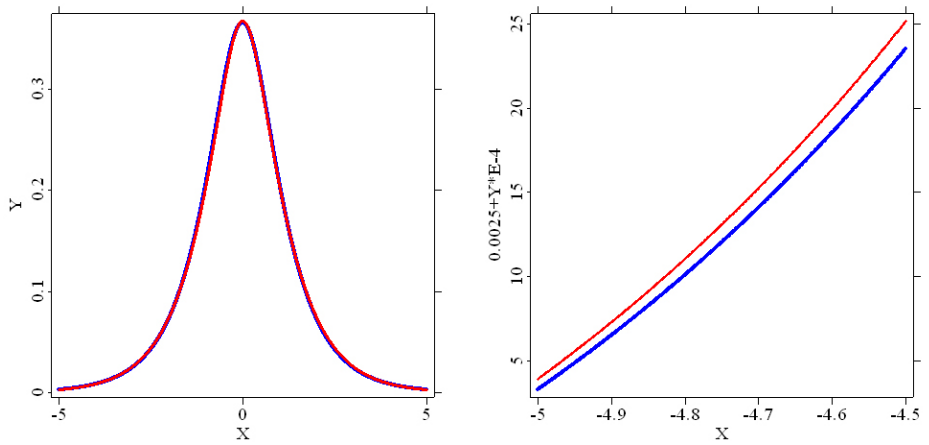

Fig. 4.19: Comparison between original (blue) and estimated (red) pdf curves with parameters (0.49,1,0,1,0) and integer λ. `simint03.xpl`

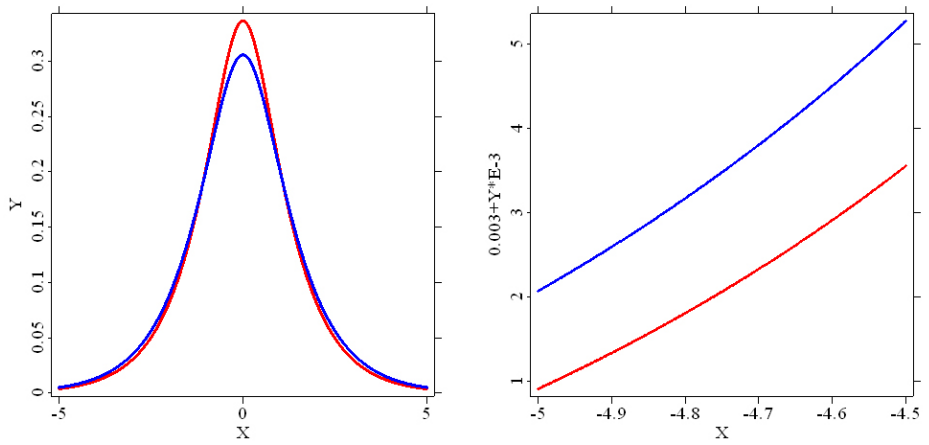

Fig. 4.20: Comparison between original (blue) and estimated (red) pdf curves with parameters (1,1,0,1,0) and integer λ.

`simint04.xpl`

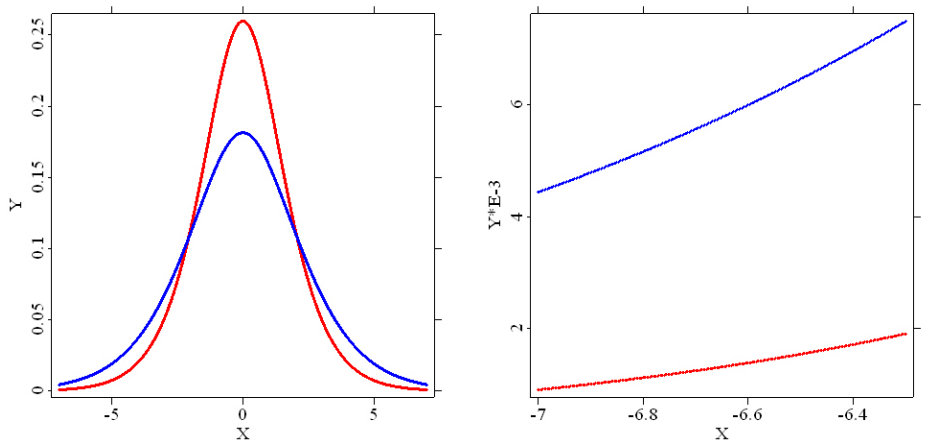

Fig. 4.21: Comparison between original (blue) and estimated (red) pdf curves with parameters (3,1,0,1,0) and integer λ.

`simint05.xpl`

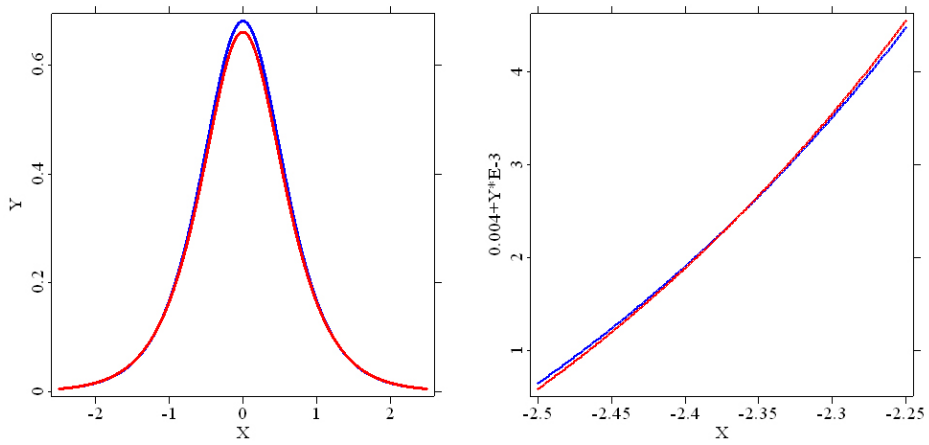

Fig. 4.22: Comparison between original (blue) and estimated (red) pdf curves with parameters (-0.5,2,0,1,0) and integer λ. simint06.xpl

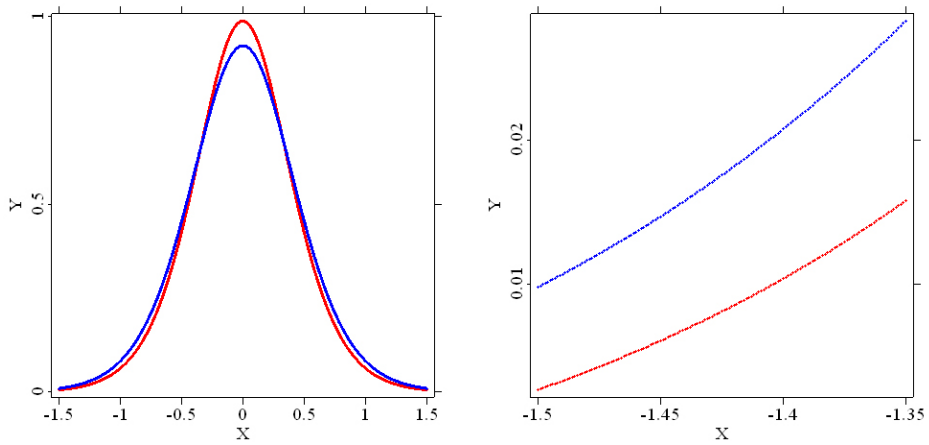

Fig. 4.23: Comparison between original (blue) and estimated (red) pdf curves with parameters (-0.5,4.5,0,1,0) and integer λ. simint07.xpl

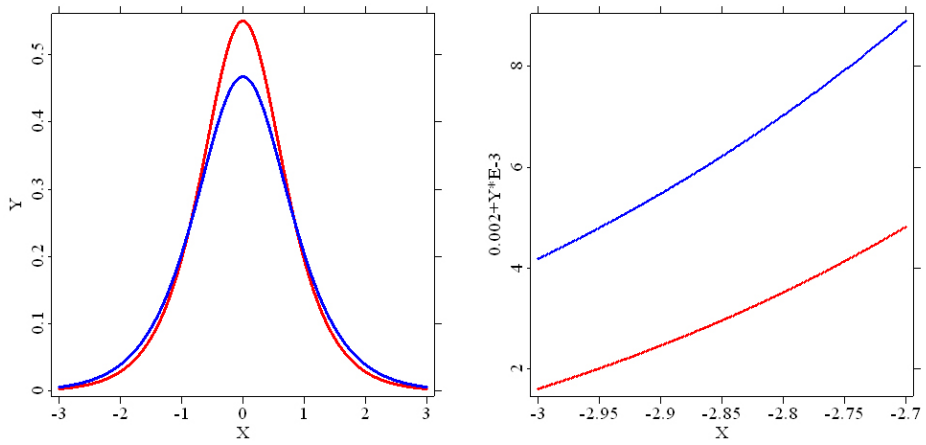

Fig. 4.24: Comparison between original (blue) and estimated (red) pdf curves with parameters (1,2,0,1,0) and integer λ.

`simint08.xpl`

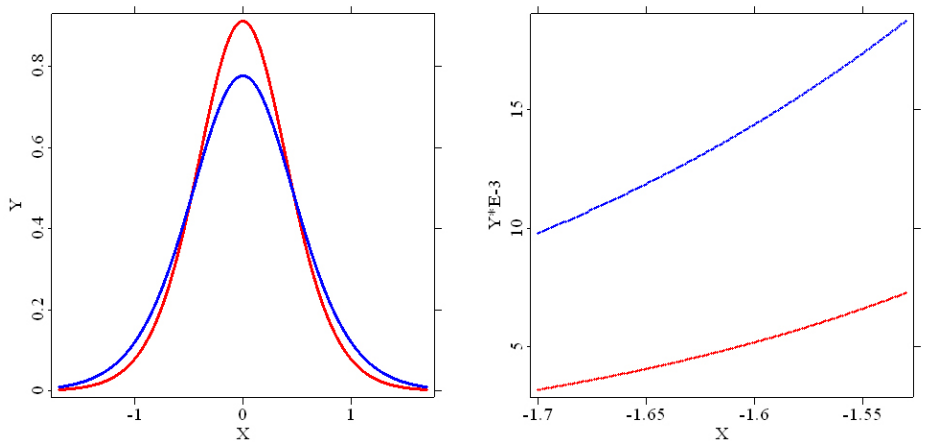

Fig. 4.25: Comparison between original (blue) and estimated (red) pdf curves with parameters (1,4.5,0,1,0) and integer λ.

`simint09.xpl`

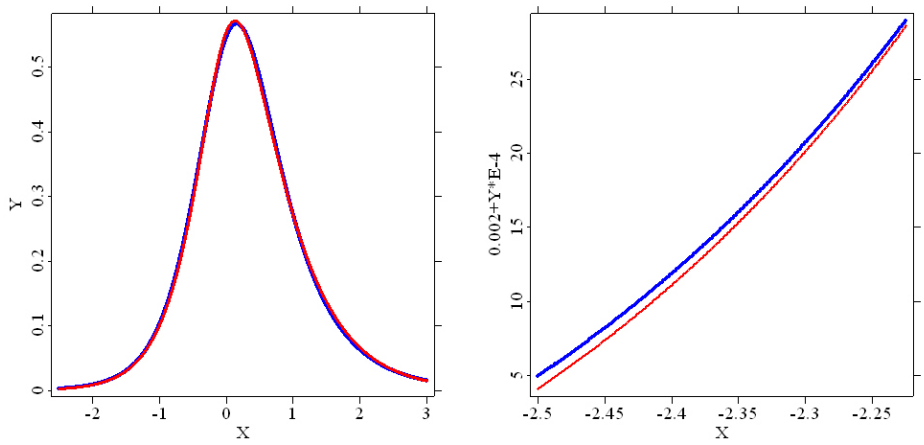

Fig. 4.26: Comparison between original (blue) and estimated (red) pdf curves with parameters (-0.5,1.5,0.5,1,0) and integer λ. `simint10.xpl`

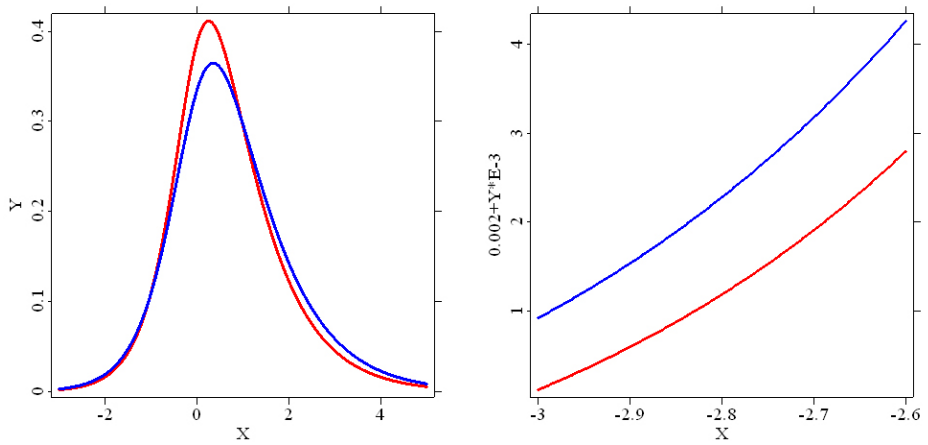

Fig. 4.27: Comparison between original (blue) and estimated (red) pdf curves with parameters (1,1.5,0.5,1,0) and integer λ. `simint11.xpl`

5. APPLICATION TO FINANCIAL MARKET

5.1 Data Description

As the first step into application to financial market, we choose three sets of data from http://www.quantlet.org/mdbase: BMW stock price, Thyssen stock price and foreign exchange rate.

The dataset of BMW stock price (BMW) contains opening prices, highest prices, lowest prices and closing prices from Jan.02,1990 to Dec.30,1992 in Federal Republic of Germany. Including the date, we have a 747×5 dataset.

Similarly, the dataset of Thyssen stock price (THY) contains opening prices, highest prices, lowest prices and closing prices from Jan.02,1990 to Dec.30,1992 in Federal Republic of Germany. Together with the date, we have also a 747×5 dataset. We use closing prices to calculate returns.

The data of foreign exchange rate studied here contain

- foreign exchange rate German Mark to US Dollar (DMUSD)
- foreign exchange rate British Pound to US Dollar (BPUSD)

They cover daily observations from Dec.01,1979 to Apr.01,1994 in Federal Republic of Germany. The dimension of the data is therefore 3720×2.

5.2 Data Transformation

As presented before, an important empirical fact in financial market is that returns of financial assets are often heavy-tail distributed. We assume the return is GH distributed and estimate the five parameters.

In order to get a stationary process, log-returns of exchange rates are used. If we denote r_t as log-return at time t,

$$r_t = \log(p_t) - \log(p_{t-1}) = \log\left(\frac{p_t}{p_{t-1}}\right) \tag{5.1}$$

where p_t is the observed price of financial asset or exchange rate of currency at time t. In the case of exchange rate, r_t represents the logarithm of the financial return at time t of holding a unit of the currency.

The KPSS test from Kwiatkowski, Philipps, Schmidt and Shin is then employed to test for stationarity. The regression model with a time trend has the form

$$X_t = c + \mu t + k \sum_{i=1}^{t} \xi_i + \eta_t \tag{5.2}$$

with stationary η_i and i.i.d. ξ_t with an expected value 0 and variance 1. For $k = 0$ the process is trend stationary, while it is an integrated process for $k \neq 0$. The null hypothesis is $H_0 : k = 0$, and the alternative hypothesis is $H_1 : k \neq 0$.

The KPSS test statistic is

$$KPSS_T = \frac{\sum_{t=1}^{n} S_t^2}{n^2 \hat{\omega}_T^2}, \tag{5.3}$$

with

$$\begin{aligned} S_t &= \sum_{i=1}^{t} \hat{\eta}_i, \\ \hat{\omega}_T^2 &= \hat{\sigma}_\eta^2 + 2 \sum_{\tau=1}^{T} \left(1 - \frac{\tau}{T-1}\right) \hat{\gamma}_\tau, \end{aligned}$$

where $\hat{\sigma}_\eta^2$ is the variance estimator of η_t and $\hat{\eta}_\tau = 1/n \sum_{t=\tau+1}^{n} \hat{\eta}_t \hat{\eta}_{t-\tau}$ is the covariance estimator.

The results of Table 5.1 indicates that the stock prices and the exchange rates are not stationary or trend stationary, since in every case the null hypothesis at a significance level of 1% is rejected. On the other hand, as Table 5.2 exhibits, all the log-returns are stationary even at the significance level of 10%. When tested with time trend, the null hypothesis is accepted at the level of 10% for the case of BMW and THY, while the process of log-return of DMUSD and BPUSD is trend stationary at the level of 1% and 5% respectively.

Now we have a stationary process, a further transformation leads us to the problem of volatility. As indicated by Franke, Härdle and Hafner (2004), volatility plays an important role in modeling financial systems and time series. Although the volatility is not observable, it can be estimated from

	Statistic	Critical Value		
		0.10	0.05	0.01
BMW without time trend	1.149	0.347	0.463	0.739
BMW with time trend	0.512	0.119	0.146	0.216
THY without time trend	2.470	0.347	0.463	0.739
THY with time trend	0.481	0.119	0.146	0.216
DMUSD without time trend	9.165	0.347	0.463	0.739
DMUSD with time trend	1.982	0.119	0.146	0.216
BPUSD without time trend	2.349	0.347	0.463	0.739
BPUSD with time trend	2.337	0.119	0.146	0.216

Tab. 5.1: KPSS test for stock prices and exchange rates with reference point $T = 8$.

	Statistic	Critical Value		
		0.10	0.05	0.01
BMW without time trend	0.086	0.347	0.463	0.739
BMW with time trend	0.082	0.119	0.146	0.216
THY without time trend	0.093	0.347	0.463	0.739
THY with time trend	0.084	0.119	0.146	0.216
DMUSD without time trend	0.322	0.347	0.463	0.739
DMUSD with time trend	0.181	0.119	0.146	0.216
BPUSD without time trend	0.176	0.347	0.463	0.739
BPUSD with time trend	0.122	0.119	0.146	0.216

Tab. 5.2: KPSS test for log-returns of stocks and currencies with reference point $T = 8$.

the data. The problem is to find an appropriate model for volatility. ARCH models are the most important class of models in this area.

ARCH models (autoregressive conditional heteroscedasticity) can efficiently model the typical empirical findings in financial time series, the conditional heteroscedasticity. After the collapse of the currency system in Bretton Woods and the following time period of flexible exchange rates in the seventies, such models were increasingly necessary for researches and practitioners. The ARCH model can be generalized by extending it with autoregressive terms of the volatility. The resulting model is called Generalized ARCH model or GARCH model, which is appropriate for modeling returns of financial assets.

The process (ε_t), $t \in \mathbb{Z}$, is GARCH(p, q), if $\mathrm{E}[\varepsilon_t | F_{t-1}] = 0$,

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_i \sigma_{t-j}^2, \tag{5.4}$$

	ω	α	β
BMW	4.7032e-05	0.13015	0.73617
THY	1.6982e-05	0.20505	0.76125
DMUSD	1.6525e-06	0.074956	0.89405
BPUSD	7.5977e-07	0.052839	0.93213

Tab. 5.3: Parameter estimates of GARCH(1,1)

We assume that the log-returns follow GARCH(1,1) process. With the help of garchest.xpl, the processes can be estimated. The parameter estimates are listed in Table 5.3.

Together with parameters, the volatilities of the processes are also estimated by garchest.xpl. To rule out the influence of volatility, the log-returns are divided by volatility estimates, which means $Z_t = \varepsilon_t/\sigma_t$ is assumed to be GH distributed.

5.3 Estimation

Table 5.4 and Figure 5.1 - 5.2 demonstrate the results of GH distribution estimation with fractional λ. The left panel of Figure 5.1 illustrates the results of the case BMW, while the right panel shows the results of the case THY. Similarly, the left and right panel of Figure 5.2 display the results of the case DMUSD and of the case BPUSD respectively. The red curves in the graphics represent the pdf curves of estimated GH distributions. The blue curves show the outcome of kernel density estimation (a nonparametric method), which is utilized here as a comparison. From the graphic, we find that two methods of estimation give rather similar pdf curves in all cases. However, from the gaps between the red and blue curves we conclude that the results are more desirable in the cases of DMUSD and BPUSD than in the cases of BMW and THY. It is probably because the former have larger sample size than the latter.

Table 5.5 and Figure 5.3 - 5.4 display the results of GH distribution estimation with integer λ. Again, we have more satisfactory results in DMUSD and BPUSD than in BMW and THY. Either in the left or in the right panel of Figure 5.4, the red curve and blue curve almost overlap, while there are relatively larger gaps around the peak areas in Figure 5.3. The comparison between Table 5.4 and 5.5 reveals that the log-likelihood values of fractional and integer cases are very close to each other. The estimation with fractional λ performs slightly better than that with integer λ in all cases, since the former has a higher log-likelihood values.

	λ	α	β	δ	μ	LogLH
BMW	-1.454790	0.441041	-0.030650	1.199405	0.009788	-994.275246
THY	-2.051983	0.149017	-0.116073	1.433777	0.065313	-991.483863
DMUSD	1.628804	1.796164	-0.024493	0.000000	0.032792	-5166.568932
BPUSD	1.711117	1.846472	0.079376	0.000000	-0.060799	-5172.000888

Tab. 5.4: Parameter estimates of GH distributions with fractional λ

	λ	α	β	δ	μ	LogLH
BMW	1.000000	1.519522	-0.019417	0.236442	-0.002284	-996.149029
THY	1.000000	1.591577	-0.079240	0.354969	0.027018	-996.386270
DMUSD	1.000000	1.613605	-0.025404	0.564662	0.033675	-5168.951861
BPUSD	1.000000	1.646890	0.074733	0.618643	-0.056103	-5174.957993

Tab. 5.5: Parameter estimates of GH distributions with integer λ

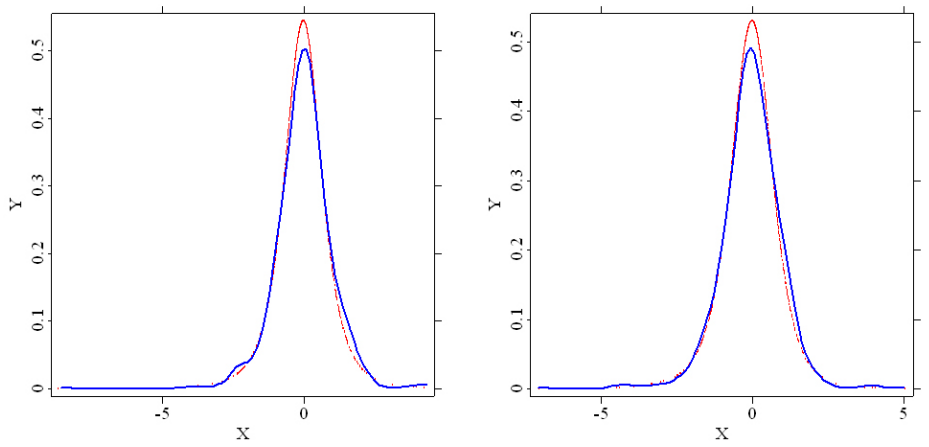

Fig. 5.1: Comparison between kernel density estimation (blue) and GH estimation with fractional λ (red). Left - BMW; Right - THY GHbt.xpl

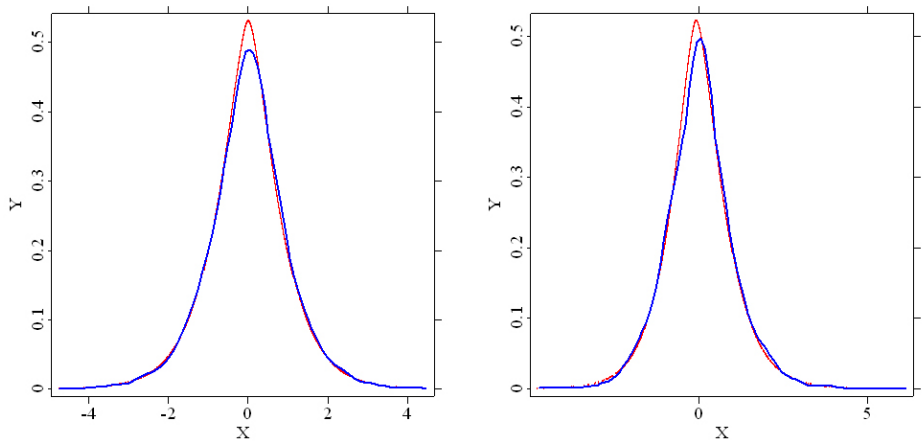

Fig. 5.2: Comparison between kernel density estimation (blue) and GH estimation with fractional λ (red). Left - DMUSD; Right - BPUSD `afm.xpl`

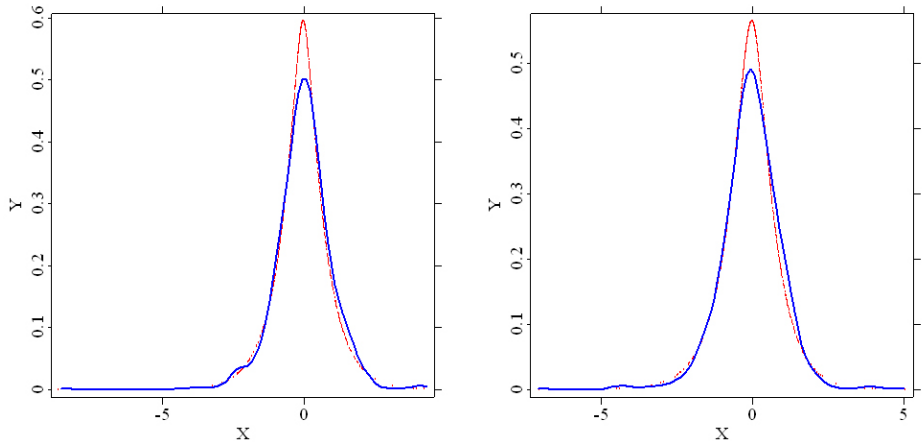

Fig. 5.3: Comparison between kernel density estimation (blue) and GH estimation with integer λ (red). Left - BMW;Right - THY `GHbtint.xpl`

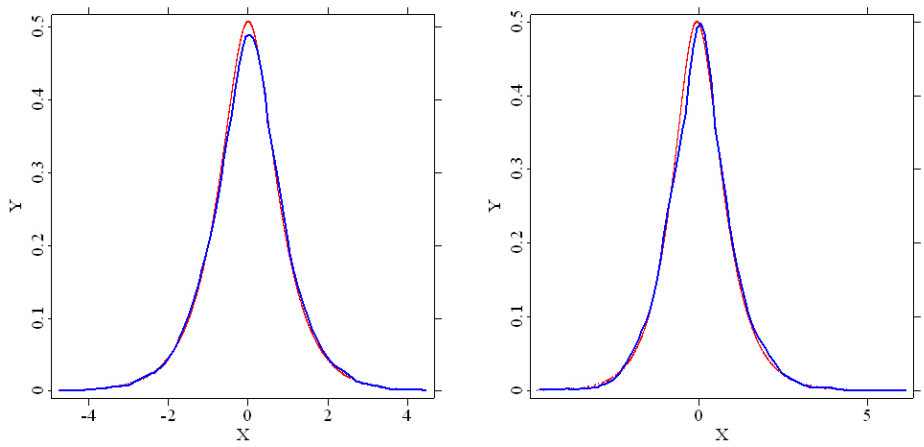

Fig. 5.4: Comparison between kernel density estimation (blue) and GH estimation with integer λ (red). Left - DMUSD;Right - BPUSD `afmint.xpl`

6. SUMMARY AND OUTLOOKS

This thesis focuses on a particular heavy-tailed distribution: GH distributions. We have looked through the major features of GH distributions: parameters, moment generating function, characteristic function and so on. Some of the subclasses and limiting distributions of GH distributions have also been examined.

The emphasis of the thesis is using maximum likelihood estimation to estimate the five parameters of GH distributions. Several numerical algorithms of searching for extreme, including Golden Section search and parabolic interpolation, are introduced and Powell's methods in multidimensions, which are based on the algorithms introduced, are utilized in our case.

The outcome of the estimation is assessed by simulation studies. Different original parameter sets are employed to measure the performance of estimation. The results of estimation with fractional λ and its simplified version, estimation with integer λ, are presented. Estimation with fractional λ performs better, while estimation with integer λ takes much less time to process.

When the estimation is exercised in the financial environment with real data, kernel density estimation is used as comparison. The results of application are rather desirable, since in every case KDE curve is very similar to GH estimated curve.

While being exceedingly well when the original parameters of GH distributions are not very large, the performances of estimation are not so desirable if the original parameters take large positive values. A useful extension of the current work would therefore involve closer examination of the reasons behide the defection of the GH estimation and possible solutions to it. Another extension would incorporate a scheme which improve the performance of estimation with integer λ or reduce the costs of estimation with fractional λ.

7. APPENDIX

7.1 C Codes

7.1.1 mlgh.c

Return the five parameter estimates of GH distribution and log-likelihood value with fractional λ.

Some of the functions used here are from Prause (1999) and the book *Numerical Recipes in C++* written by Press et al. (2002).

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.141592653589793116
#define ACC 40.0
#define TINY 1.0e-20
#define ITMAX 3000
#define MAXIT 10000

#define TOL 2.0e-4
#define NR_END 1
#define FREE_ARG char*
#define CGOLD 0.3819660
#define ZEPS 1.0e-10
#define SHFT(a,b,c,d) (a)=(b);(b)=(c);(c)=(d);
#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))
#define GOLD 1.618034
#define GLIMIT 100.0
#define EPS 1.0e-10
#define XMIN 2.0
#define FPMIN 1.0e-30

static double maxarg1,maxarg2;

#define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (
  maxarg2) ?\
        (maxarg1) : (maxarg2))

static double sqrarg;

```

```
31 #define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
32
33 void mlgh(double *lambda, double *alpha, double *beta, double
     *delta, double *mu, double *ml);
34 double logligh(double *p);
35
36 void bessik (double x, double xnu, double *ri, double *rk,
     double *rip, double *rkp);
37 void beschb (double x, double *gam1, double *gam2, double *
     gampl, double *gammi);
38 double chebev(double a, double b, double c[], int m, double x
     );
39 double *vector(long nl, long nh);
40 double **matrix(long nrl, long nrh, long ncl, long nch);
41 void free_vector(double *v, long nl, long nh);
42 void free_matrix(double **m, long nrl, long nrh, long ncl,
     long nch);
43 void powell(double p[], double **xi, int n, double ftol, int
     *iter, double *fret, double (*func)(double []));
44 void linmin(double p[], double xi[], int n, double *fret,
     double (*func)(double []));
45 double f1dim(double x);
46 double brent(double ax, double bx, double cx, double (*f)(
     double), double tol, double *xmin);
47 void mnbrak(double *ax, double *bx, double *cx, double *fa,
     double *fb, double *fc, double (*func)(double));
48
49 double *readdata(int *n);
50
51 FILE *DATA;
52
53 double *X;
54 int H;
55
56 void main()
57 {
58     int n,i;
59     double *r, lambda, alpha, beta, delta, mu, ml;
60     r = readdata(&n);
61     H = n;
62     X = malloc(sizeof(double)*(n+1));
63     for (i = 1; i <= H; i++) *(X+i)=*(r+i);
64     mlgh(&lambda, &alpha, &beta, &delta, &mu, &ml);
65     printf("%9.16f\n%9.16f\n%9.16f\n%9.16f\n%9.16f\n\n%9.16f\
       n", lambda, alpha, beta, delta, mu, ml);
66 }
67
68 double *readdata(int *n)
69 {
70     int i;
71     double *r, tmp;
72
73     DATA = fopen("data.txt", "r");
74     fscanf(DATA, "%lf", &tmp);
```

```
75      *n = (int)tmp;
76      r = malloc(sizeof(double)*((*n)+1));
77      *r = 0.0;
78      for ( i = 1 ; i <= (*n) ; i ++ ) {
79          fscanf(DATA, "%lf", &tmp);
80          *(r+i) = tmp;
81      }
82      fclose(DATA);
83      return r;
84  }
85
86
87  void mlgh(double *lambda, double *alpha, double *beta, double
        *delta, double *mu, double *ml)
88  {
89      int iter;
90      long i;
91      double *p, **xi, ftol=1.0e-6, fret, psi, chi, t1, t2, ri,
           rk, rip, rkp, tmp, dif, a1, a2;
92
93      p = vector(1,5);
94      xi = matrix(1,5,1,5);
95
96      p[1] = 0.0; p[2] = 0.0; p[3] = 0.0; p[4] = 0.0; p[5] =
          0.0;
97
98      xi[1][1] = 1.0; xi[1][2] = 0.0; xi[1][3] = 0.0; xi[1][4]
          = 0.0; xi[1][5] = 0.0;
99      xi[2][1] = 0.0; xi[2][2] = 1.0; xi[2][3] = 0.0; xi[2][4]
          = 0.0; xi[2][5] = 0.0;
100     xi[3][1] = 0.0; xi[3][2] = 0.0; xi[3][3] = 1.0; xi[3][4]
          = 0.0; xi[3][5] = 0.0;
101     xi[4][1] = 0.0; xi[4][2] = 0.0; xi[4][3] = 0.0; xi[4][4]
          = 1.0; xi[4][5] = 0.0;
102     xi[5][1] = 0.0; xi[5][2] = 0.0; xi[5][3] = 0.0; xi[5][4]
          = 0.0; xi[5][5] = 1.0;
103
104     powell(p, xi, 5, ftol, &iter, &fret, &logligh);
105
106     *lambda = p[1];
107     *alpha = sqrt(p[3]*p[3]+exp(p[2]));
108     *beta = p[3];
109     *delta = exp(0.5*p[4]);
110     *mu = p[5];
111
112     psi = p[2];
113     chi = p[4];
114     t1=0;
115     t2=0;
116     for ( i = 1 ; i <= H ; i ++ ){
117         dif = *(X+i)-*mu;
118         a1 = exp(chi)+dif*dif;
119         t1 = t1+log(a1);
120         a2 = sqrt((*beta*(*beta)+exp(psi))*a1);
```

```
121         bessik(a2, fabs(*lambda-0.5), &ri, &rk, &rip, &rkp);
122         t2 = t2+log(rk)+ *beta*dif;
123     }
124     tmp = exp(0.5*(psi+chi));
125     bessik(tmp, fabs(*lambda), &ri, &rk, &rip, &rkp);
126     tmp = log(rk);
127     tmp = 0.5*(*lambda)*psi-0.5*log(2*PI)-(0.5*(*lambda)
          -0.25)*log(*beta*(*beta)+exp(psi))-0.5*(*lambda)*chi-tmp
          ;
128     tmp = H*tmp+(0.5*(*lambda)-0.25)*t1+t2;
129
130     *ml = tmp;
131     free_vector(p,1,5);
132     free_matrix(xi,1,5,1,5);
133 }
134
135 double logligh(double p[])
136 {
137     long i;
138     double a1, a2, dif, lambda, psi, beta, chi, mu, t1, t2,
          tmp, ri, rk, rip, rkp;
139
140     lambda = p[1];
141     psi = p[2];
142     beta = p[3];
143     chi = p[4];
144     mu = p[5];
145     t1=0;
146     t2=0;
147     for ( i = 1 ; i <= H ; i ++ ){
148         dif = *(X+i)-mu;
149         a1 = exp(chi)+dif*dif;
150         t1 = t1+log(a1);
151         a2 = sqrt((beta*beta+exp(psi))*a1);
152         bessik(a2, fabs(lambda-0.5), &ri, &rk, &rip, &rkp);
153         t2 = t2+log(rk)+beta*dif;
154     }
155     tmp = exp(0.5*(psi+chi));
156     bessik(tmp, fabs(lambda), &ri, &rk, &rip, &rkp);
157     tmp = log(rk);
158     tmp = 0.5*lambda*psi-0.5*log(2*PI)-(0.5*lambda-0.25)*log(
          beta*beta+exp(psi))-0.5*lambda*chi-tmp;
159     tmp = H*tmp+(0.5*lambda-0.25)*t1+t2;
160     tmp = -tmp;
161
162     return tmp;
163 }
164
165
166 void bessik (double x, double xnu, double *ri, double *rk,
      double *rip, double *rkp)
167 {
168     double a,a1,b,c,d,del,del1,delh,dels,e,f,fact,fact2,ff,
          gam1,gam2,gammi,gampl,h,p,pimu,q,q1,q2,qnew,ril,
```

```
169         ril1,rimu,rip1,ripl,ritemp,rk1,rkmu,rkmup,rktemp,s,
              sum,sum1,x2,xi,xi2,xmu,xmu2;
170     int i,l,nl;
171
172
173     if (x <= 0.0 || xnu < 0.0)
174     {
175         printf("bad arguments in bessik");
176         exit(EXIT_FAILURE);
177     }
178     nl = (int)(xnu+0.5);
179     xmu = xnu-nl;
180     xmu2 = xmu*xmu;
181     xi = 1.0/x;
182     xi2 = 2.0*xi;
183     h = xnu*xi;
184     if (h < FPMIN) h = FPMIN;
185     b = xi2*xnu;
186     d = 0.0;
187     c = h;
188     for (i=0;i<MAXIT;i++) {
189         b += xi2;
190         d = 1.0/(b+d);
191         c = b+1.0/c;
192         del = c*d;
193         h = del*h;
194         if (fabs(del-1.0) <= EPS) break;
195     }
196     if (i >= MAXIT)
197     {
198         printf("x too large in bessik; try asymptotic
              expansion");
199         exit(EXIT_FAILURE);
200     }
201     ril = FPMIN;
202     ripl = h*ril;
203     ril1 = ril;
204     rip1 = ripl;
205     fact = xnu*xi;
206     for (l = nl-1; l >= 0; l--) {
207         ritemp = fact*ril+ripl;
208         fact -= xi;
209         ripl = fact*ritemp + ril;
210         ril = ritemp;
211     }
212     f = ripl/ril;
213     if (x < XMIN) {
214         x2 = 0.5*x;
215         pimu = PI*xmu;
216         fact = (fabs(pimu) < EPS ? 1.0 : pimu/sin(pimu));
217         d = -log(x2);
218         e = xmu*d;
219         fact2 = (fabs(e) < EPS ? 1.0 : sinh(e)/e);
220         beschb(xmu,&gam1,&gam2,&gampl,&gammi);
```

```
221         ff = fact*(gam1*cosh(e)+gam2*fact2*d);
222         sum = ff;
223         e = exp(e);
224         p = 0.5*e/gampl;
225         q = 0.5/(e*gammi);
226         c = 1.0;
227         d = x2*x2;
228         sum1 = p;
229         for (i=1; i<=MAXIT;i++) {
230             ff = (i*ff+p+q)/(i*i-xmu2);
231             c *= (d/i);
232             p /= (i - xmu);
233             q /= (i + xmu);
234             del = c*ff;
235             sum += del;
236             del1 = c*(p-i*ff);
237             sum1 += del1;
238             if (fabs(del) < fabs(sum)*EPS) break;
239         }
240         if (i > MAXIT)
241         {
242         printf("bessik series failed to converge");
243         exit(EXIT_FAILURE);
244         }
245         rkmu = sum;
246         rk1 = sum1*xi2;
247     }
248     else {
249             b = 2.0*(1.0+x);
250             d = 1.0/b;
251             h = delh = d;
252             q1 = 0.0;
253             q2 = 1.0;
254             a1 = 0.25 - xmu2;
255             q = c = a1;
256             a = -a1;
257             s = 1.0+q*delh;
258             for (i=1; i<MAXIT;i++) {
259                 a -= 2*i;
260                 c = -a*c/(i+1.0);
261                 qnew = (q1-b*q2)/a;
262                 q1 = q2;
263                 q2 = qnew;
264                 q += c*qnew;
265                 b += 2.0;
266                 d = 1.0/(b+a*d);
267                 delh = (b*d-1.0)*delh;
268                 h += delh;
269                 dels = q*delh;
270                 s += dels;
271                 if (fabs(dels/s)<=EPS) break;
272             }
273             if (i  >= MAXIT)
274             {
```

```
275                 printf("bessik: failure to converge in cf2");
276                 exit(EXIT_FAILURE);
277             }
278             h = a1*h;
279             rkmu = sqrt(PI/(2.0*x))*exp(-x)/s;
280             rk1 = rkmu*(xmu+x+0.5-h)*xi;
281     }
282     rkmup = xmu*xi*rkmu-rk1;
283     rimu = xi/(f*rkmu-rkmup);
284     *ri = (rimu*ril1)/ril;
285     *rip = (rimu*rip1)/ril;
286     for (i=1;i <= nl;i++) {
287         rktemp = (xmu+i)*xi2*rk1+rkmu;
288         rkmu = rk1;
289         rk1 = rktemp;
290     }
291     *rk = rkmu;
292     *rkp = xnu*xi*rkmu-rk1;
293 }
294
295 void beschb (double x, double *gam1, double *gam2, double *
      gampl, double *gammi)
296 {
297     int NUSE1 = 7, NUSE2 = 8;
298     double xx, *c1, *c2;
299
300     c1 = vector(1,7);
301     c2 = vector (1,8);
302
303     c1[1] = -1.142022680371168e0;
304     c1[2] = 6.5165112670737e-3;
305     c1[3] = 3.087090173086e-4;
306     c1[4] = -3.4706269649e-6;
307     c1[5] = 6.9437664e-9;
308     c1[6] = 3.67795e-11;
309     c1[7] = -1.356e-13;
310
311     c2[1] = 1.843740587300905e0;
312     c2[2] = -7.68528408447867e-2;
313     c2[3] = 1.2719271366546e-3;
314     c2[4] = -4.9717367042e-6;
315     c2[5] = -3.31261198e-8;
316     c2[6] = 2.423096e-10;
317     c2[7] = -1.702e-13;
318     c2[8] = -1.49e-15;
319
320     xx = 8.0*x*x-1.0;
321     *gam1 = chebev(-1.0,1.0,c1,NUSE1,xx);
322     *gam2 = chebev(-1.0,1.0,c2,NUSE2,xx);
323     *gampl = *gam2-x*(*gam1);
324     *gammi = *gam2+x*(*gam1);
325     free_vector(c1,1,7);
326     free_vector(c2,1,8);
327 }
```

```
328
329 double chebev(double a, double b, double c[], int m, double x
      )
330 {
331     double d = 0.0, dd = 0.0, sv,y,y2;
332     int j;
333
334     if ((x-a)*(x-b) > 0.0)
335     {
336         printf("x not in routine chebev");
337         exit(EXIT_FAILURE);
338     }
339     y2 = 2.0*(y = (2.0*x-a-b)/(b-a));
340     for (j=m;j>1;j--) {
341         sv = d;
342         d = y2*d-dd+c[j];
343         dd = sv;
344     }
345     return y*d-dd+0.5*c[1];
346 }
347
348
349 double *vector(long nl, long nh)
350 {
351     double *v;
352
353     v = (double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(
          double)));
354
355     return v-nl+NR_END;
356 }
357
358 double **matrix(long nrl, long nrh, long ncl, long nch)
359 {
360     long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
361     double **m;
362
363     m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double
          *)));
364     m += NR_END;
365     m -= nrl;
366     m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*
          sizeof(double)));
367     m[nrl] += NR_END;
368     m[nrl] -= ncl;
369     for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;
370
371     return m;
372 }
373
374 void free_vector(double *v, long nl, long nh)
375 {
376     free((FREE_ARG) (v+nl-NR_END));
377 }
```

```
378
379 void free_matrix(double **m, long nrl, long nrh, long ncl,
      long nch)
380 {
381     free((FREE_ARG) (m[nrl]+ncl-NR_END));
382     free((FREE_ARG) (m+nrl-NR_END));
383 }
384
385 void powell(double p[], double **xi, int n, double ftol, int
      *iter, double *fret, double (*func)(double []))
386 {
387     int i,ibig,j;
388     double del,fp,fptt,t,*pt,*ptt,*xit;
389
390     pt=vector(1,n);
391     ptt=vector(1,n);
392     xit=vector(1,n);
393     *fret=(*func)(p);
394     for (j=1;j<=n;j++) pt[j]=p[j];
395     for (*iter=1;;++(*iter)) {
396         fp=(*fret);
397         ibig=0;
398         for (i=1;i<=n;i++) {
399             for (j=1;j<=n;j++) xit[j]=xi[j][i];
400             fptt=(*fret);
401             linmin(p,xit,n,fret,func);
402             if (fptt-(*fret) > del) {
403                 del=fptt-(*fret);
404                 ibig=i;
405             }
406         }
407         if (2.0*(fp-(*fret)) <= ftol*(fabs(fp)+fabs(*fret))+
              TINY) {
408             free_vector(xit,1,n);
409             free_vector(ptt,1,n);
410             free_vector(pt,1,n);
411             return;
412         }
413         if (*iter == ITMAX) {
414             printf("powell exceeding maximum iterations.");
415             exit(EXIT_FAILURE);
416         }
417         for (j=1;j<=n;j++) {
418             ptt[j]=2.0*p[j]-pt[j];
419             xit[j]=p[j]-pt[j];
420             pt[j]=p[j];
421         }
422         fptt=(*func)(ptt);
423         if (fptt < fp) {
424             t=2.0*(fp-2.0*(*fret)+fptt)*SQR(fp-(*fret)-del)-
                del*SQR(fp-fptt);
425             if (t < 0.0) {
426                 linmin(p,xit,n,fret,func);
427                 for (j=1;j<=n;j++) {
```

```
428                     xi[j][ibig]=xi[j][n];
429                     xi[j][n]=xit[j];
430                 }
431             }
432         }
433     }
434 }
435
436 int ncom;
437 double *pcom, *xicom, (*nrfunc)(double []);
438
439 void linmin(double p[], double xi[], int n, double *fret,
      double (*func)(double []))
440 {
441     int j;
442     double xx,xmin,fx,fb,fa,bx,ax;
443
444     ncom=n;
445     pcom=vector(1,n);
446     xicom=vector(1,n);
447     nrfunc=func;
448     for (j=1;j<=n;j++) {
449         pcom[j]=p[j];
450         xicom[j]=xi[j];
451     }
452     ax=0.0;
453     xx=1.0;
454     mnbrak(&ax,&xx,&bx,&fa,&fx,&fb,f1dim);
455     *fret=brent(ax,xx,bx,f1dim,TOL,&xmin);
456     for (j=1;j<=n;j++) {
457         xi[j] *= xmin;
458         p[j] += xi[j];
459     }
460     free_vector(xicom,1,n);
461     free_vector(pcom,1,n);
462 }
463
464 double f1dim(double x)
465 {
466     int j;
467     double f,*xt;
468
469     xt=vector(1,ncom);
470     for (j=1;j<=ncom;j++) xt[j]=pcom[j]+x*xicom[j];
471     f=(*nrfunc)(xt);
472     free_vector(xt,1,ncom);
473     return f;
474 }
475
476 double brent(double ax, double bx, double cx, double (*f)(
      double), double tol, double *xmin)
477 {
478     int iter;
479     double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm
```

```
    ;
480 double e=0.0;
481
482 a=(ax < cx ? ax : cx);
483 b=(ax > cx ? ax : cx);
484 x=w=v=bx;
485 fw=fv=fx=(*f)(x);
486 for (iter=1;iter<=ITMAX;iter++) {
487     xm=0.5*(a+b);
488     tol2=2.0*(tol1=tol*fabs(x)+ZEPS);
489     if (fabs(x-xm) <= (tol2-0.5*(b-a))) {
490         *xmin=x;
491         return fx;
492     }
493     if (fabs(e) > tol1) {
494         r=(x-w)*(fx-fv);
495         q=(x-v)*(fx-fw);
496         p=(x-v)*q-(x-w)*r;
497         q=2.0*(q-r);
498         if (q > 0.0) p = -p;
499         q=fabs(q);
500         etemp=e;
501         e=d;
502         if (fabs(p) >= fabs(0.5*q*etemp) || p <= q*(a-x)
              || p >= q*(b-x))
503             d=CGOLD*(e=(x >= xm ? a-x : b-x));
504         else {
505             d=p/q;
506             u=x+d;
507             if (u-a < tol2 || b-u < tol2)
508                 d=SIGN(tol1,xm-x);
509         }
510     }
511     else {
512         d=CGOLD*(e=(x >= xm ? a-x : b-x));
513     }
514     u=(fabs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
515     fu=(*f)(u);
516     if (fu <= fx) {
517         if (u >= x) a=x; else b=x;
518         SHFT(v,w,x,u)
519         SHFT(fv,fw,fx,fu)
520     }
521     else {
522         if (u < x) a=u; else b=u;
523         if (fu <= fw || w == x) {
524             v=w;
525             w=u;
526             fv=fw;
527             fw=fu;
528         }
529         else if (fu <= fv || v == x || v == w) {
530             v=u;
531             fv=fu;
```

```
532             }
533         }
534     }
535     printf("Too many iterations in brent");
536     exit(EXIT_FAILURE);
537 }
538
539 void mnbrak(double *ax, double *bx, double *cx, double *fa,
      double *fb, double *fc, double (*func)(double))
540 {
541     double ulim,u,r,q,fu,dum;
542
543     *fa=(*func)(*ax);
544     *fb=(*func)(*bx);
545     if (*fb > *fa) {
546         SHFT(dum,*ax,*bx,dum)
547         SHFT(dum,*fb,*fa,dum)
548     }
549     *cx=(*bx)+GOLD*(*bx-*ax);
550     *fc=(*func)(*cx);
551     while (*fb > *fc) {
552         r=(*bx-*ax)*(*fb-*fc);
553         q=(*bx-*cx)*(*fb-*fa);
554         u=(*bx)-((*bx-*cx)*q-(*bx-*ax)*r)/(2.0*SIGN(FMAX(fabs
              (q-r),TINY),q-r));
555         ulim=(*bx)+GLIMIT*(*cx-*bx);
556         if ((*bx-u)*(u-*cx) > 0.0) {
557             fu=(*func)(u);
558             if (fu < *fc) {
559                 *ax=(*bx);
560                 *bx=u;
561                 *fa=(*fb);
562                 *fb=fu;
563                 return;
564             }
565             else if (fu > *fb) {
566                 *cx=u;
567                 *fc=fu;
568                 return;
569             }
570             u=(*cx)+GOLD*(*cx-*bx);
571             fu=(*func)(u);
572         }
573         else if ((*cx-u)*(u-ulim) > 0.0) {
574             fu=(*func)(u);
575             if (fu < *fc) {
576                 SHFT(*bx,*cx,u,*cx+GOLD*(*cx-*bx))
577                 SHFT(*fb,*fc,fu,(*func)(u))
578             }
579         }
580         else if ((u-ulim)*(ulim-*cx) >= 0.0) {
581             u=ulim;
582             fu=(*func)(u);
583         }
```

```
584         else {
585             u=(*cx)+GOLD*(*cx-*bx);
586             fu=(*func)(u);
587         }
588         SHFT(*ax,*bx,*cx,u)
589         SHFT(*fa,*fb,*fc,fu)
590     }
591 }
```

7.1.2 *mlghint.c*

Return the five parameter estimates of GH distribution and log-likelihood value with integer λ.

Some of the functions used here are from Prause (1999) and the book *Numerical Recipes in C++* written by Press et al. (2002).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #define PI 3.141592653589793116
6  #define ACC 40.0
7  #define TINY 1.0e-20
8  #define ITMAX 3000
9  #define MAXIT 10000
10
11 #define TOL 2.0e-4
12 #define NR_END 1
13 #define FREE_ARG char*
14 #define CGOLD 0.3819660
15 #define ZEPS 1.0e-10
16 #define SHFT(a,b,c,d) (a)=(b);(b)=(c);(c)=(d);
17 #define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))
18 #define GOLD 1.618034
19 #define GLIMIT 100.0
20 #define EPS 1.0e-10
21 #define XMIN 2.0
22 #define FPMIN 1.0e-30
23
24 static double maxarg1,maxarg2;
25
26 #define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (
     maxarg2) ?\
27         (maxarg1) : (maxarg2))
28
29 static double sqrarg;
30
31 #define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
32
33 void mlgh(int *lambda, double *alpha, double *beta, double *
     delta, double *mu, double *ml);
```

```
34 double logligh(double *p);
35
36 double besskh(int n, double x);
37 double bessk(int n, double x);
38 double bessi0(double x);
39 double bessk0(double x);
40 double bessi1(double x);
41 double bessk1(double x);
42 int ff(double x);
43 double *vector(long nl, long nh);
44 double **matrix(long nrl, long nrh, long ncl, long nch);
45 void free_vector(double *v, long nl, long nh);
46 void free_matrix(double **m, long nrl, long nrh, long ncl,
     long nch);
47 void powell(double p[], double **xi, int n, double ftol, int
     *iter, double *fret, double (*func)(double []));
48 void linmin(double p[], double xi[], int n, double *fret,
     double (*func)(double []));
49 double f1dim(double x);
50 double brent(double ax, double bx, double cx, double (*f)(
     double), double tol, double *xmin);
51 void mnbrak(double *ax, double *bx, double *cx, double *fa,
     double *fb, double *fc, double (*func)(double));
52
53 double *readdata(int *n);
54
55 FILE *DATA;
56 FILE *RES;
57
58 double *X;
59 int H;
60
61 void main()
62 {
63     int n,i,lambda;
64     double *r, alpha, beta, delta, mu, ml;
65     r = readdata(&n);
66     H = n;
67     X = malloc(sizeof(double)*(n+1));
68     for (i = 1; i <= H; i++) *(X+i)=*(r+i);
69     mlgh(&lambda, &alpha, &beta, &delta, &mu, &ml);
70     printf("%d\n%9.4f\n%9.4f\n%9.4f\n%9.4f\n\n%9.4f\n",
       lambda, alpha, beta, delta, mu, ml);
71     RES = fopen("result.txt","w");
72     fprintf(RES, "%d\n%9.4f\n%9.4f\n%9.4f\n%9.4f\n\n%9.4f\n",
        lambda, alpha, beta, delta, mu, ml);
73     fclose(RES);
74 }
75
76 double *readdata(int *n)
77 {
78     int i;
79     double *r, tmp;
80
```

```
81      DATA = fopen("data.txt", "r");
82      fscanf(DATA, "%lf", &tmp);
83      *n = (int)tmp;
84      r = malloc(sizeof(double)*((*n)+1));
85      *r = 0.0;
86      for ( i = 1 ; i <= (*n) ; i ++ ) {
87          fscanf(DATA, "%lf", &tmp);
88          *(r+i) = tmp;
89      }
90      fclose(DATA);
91      return r;
92  }
93
94
95  void mlgh(int *lambda, double *alpha, double *beta, double *
      delta, double *mu, double *ml)
96  {
97      int iter;
98      long i;
99      double *p, **xi, ftol=1.0e-6, fret, psi, chi, t1, t2, tmp
          , dif, a1, a2;
100
101     p = vector(1,5);
102     xi = matrix(1,5,1,5);
103
104     p[1] = 0.0; p[2] = 0.0; p[3] = 0.0; p[4] = 0.0; p[5] =
          0.0;
105
106     xi[1][1] = 1.0; xi[1][2] = 0.0; xi[1][3] = 0.0; xi[1][4]
          = 0.0; xi[1][5] = 0.0;
107     xi[2][1] = 0.0; xi[2][2] = 1.0; xi[2][3] = 0.0; xi[2][4]
          = 0.0; xi[2][5] = 0.0;
108     xi[3][1] = 0.0; xi[3][2] = 0.0; xi[3][3] = 1.0; xi[3][4]
          = 0.0; xi[3][5] = 0.0;
109     xi[4][1] = 0.0; xi[4][2] = 0.0; xi[4][3] = 0.0; xi[4][4]
          = 1.0; xi[4][5] = 0.0;
110     xi[5][1] = 0.0; xi[5][2] = 0.0; xi[5][3] = 0.0; xi[5][4]
          = 0.0; xi[5][5] = 1.0;
111
112     powell(p, xi, 5, ftol, &iter, &fret, &logligh);
113
114     *lambda = ff(p[1]);
115     *alpha = sqrt(p[3]*p[3]+exp(p[2]));
116     *beta = p[3];
117     *delta = exp(0.5*p[4]);
118     *mu = p[5];
119
120     psi = p[2];
121     chi = p[4];
122     t1=0;
123     t2=0;
124     for ( i = 1 ; i <= H ; i ++ ){
125         dif = *(X+i)-*mu;
126         a1 = exp(chi)+dif*dif;
```

```
127         t1 = t1+log(a1);
128         a2 = sqrt((*beta*(*beta)+exp(psi))*a1);
129         t2 = t2+log(besskh((*lambda-1),a2))+ *beta*dif;
130     }
131     tmp = exp(0.5*(psi+chi));
132     tmp = log(bessk(*lambda,tmp));
133     tmp = 0.5*(*lambda)*psi-0.5*log(2*PI)-(0.5*(*lambda)
          -0.25)*log(*beta*(*beta)+exp(psi))-0.5*(*lambda)*chi-tmp
          ;
134     tmp = H*tmp+(0.5*(*lambda)-0.25)*t1+t2;
135
136     *ml = tmp;
137     free_vector(p,1,5);
138     free_matrix(xi,1,5,1,5);
139 }
140
141 double logligh(double p[])
142 {
143     int lambda;
144     long i;
145     double a1, a2, dif, psi, beta, chi, mu, t1, t2, tmp;
146
147     lambda = ff(p[1]);
148     psi = p[2];
149     beta = p[3];
150     chi = p[4];
151     mu = p[5];
152     t1=0;
153     t2=0;
154     for ( i = 1 ; i <= H ; i ++ ){
155         dif = *(X+i)-mu;
156         a1 = exp(chi)+dif*dif;
157         t1 = t1+log(a1);
158         a2 = sqrt((beta*beta+exp(psi))*a1);
159         t2 = t2+log(besskh(lambda-1,a2))+beta*dif;
160     }
161     tmp = exp(0.5*(psi+chi));
162     tmp = log(bessk(lambda,tmp));
163     tmp = 0.5*lambda*psi-0.5*log(2*PI)-(0.5*lambda-0.25)*log(
          beta*beta+exp(psi))-0.5*lambda*chi-tmp;
164     tmp = H*tmp+(0.5*lambda-0.25)*t1+t2;
165     tmp = -tmp;
166
167     return tmp;
168 }
169
170 double besskh(int n, double x)
171 {
172     int i,j;
173     double sum1,sum2,sum3,tmp;
174     if (n <= -2) n = (int)fabs(n)-1;
175     if (n == -1 || n == 0) {
176         tmp = sqrt(PI/2)*exp(-0.5*log(x)-x);
177         return tmp;
```

```
178     }
179     tmp = 0;
180     for (i=1;i<=n;i++){
181         sum1 = 1;
182         sum2 = 1;
183         sum3 = 1;
184         for (j=1;j<=(n+i);j++){
185             sum1 = sum1*j;
186         }
187         for (j=1;j<=(n-i);j++){
188             sum2 = sum2*j;
189         }
190         for (j=1;j<=i;j++){
191             sum3 = sum3*j;
192         }
193         tmp = tmp + sum1/sum2/sum3*exp(-i*log(2*x));
194     }
195     tmp = tmp + 1;
196     tmp = sqrt(PI/2)*exp(-0.5*log(x)-x)*tmp;
197 
198     return tmp;
199 }
200 
201 double bessk(int n, double x)
202 {
203     int j;
204     double bk,bkm,bkp,tox;
205 
206     n = abs(n);
207     tox=2.0/x;
208     bkm=bessk0(x);
209     bk=bessk1(x);
210 
211     for ( j = 1 ; j < n ; j++ ) {
212         bkp=bkm+j*tox*bk;
213         bkm=bk;
214         bk=bkp;
215     }
216 
217     return bk;
218 }
219 
220 
221 double bessi0(double x)
222 {
223     double ax,ans;
224     double y;
225     if ((ax=fabs(x)) < 3.75) {
226     y=x/3.75;
227     y*=y;
228     ans=1.0+y*(3.5156229+y*(3.0899424+y*(1.2067492
229     +y*(0.2659732+y*(0.360768e-1+y*0.45813e-2)))));
230     } else {
231         y=3.75/ax;
```

```
232         ans=(exp(ax)/sqrt(ax))*(0.39894228+y*(0.1328592e-1
233         +y*(0.225319e-2+y*(-0.157565e-2+y*(0.916281e-2
234         +y*(-0.2057706e-1+y*(0.2635537e-1+y*(-0.1647633e-1
235         +y*0.392377e-2)))))));
236     }
237 return ans;
238 }
239
240 double bessk0(double x)
241 {
242     double y,ans;
243     if (x <= 2.0) {
244     y=x*x/4.0;
245     ans=(-log(x/2.0)*bessi0(x))+(-0.57721566+y*(0.42278420
246     +y*(0.23069756+y*(0.3488590e-1+y*(0.262698e-2
247     +y*(0.10750e-3+y*0.74e-5))))));
248     } else {
249         y=2.0/x;
250         ans=(exp(-x)/sqrt(x))*(1.25331414+y*(-0.7832358e-1
251         +y*(0.2189568e-1+y*(-0.1062446e-1+y*(0.587872e-2
252         +y*(-0.251540e-2+y*0.53208e-3))))));
253     }
254 return ans;
255 }
256
257 double bessi1(double x)
258 {
259     double ax,ans;
260     double y;
261     if ((ax=fabs(x)) < 3.75) {
262     y=x/3.75;
263     y*=y;
264     ans=ax*(0.5+y*(0.87890594+y*(0.51498869+y*(0.15084934
265     +y*(0.2658733e-1+y*(0.301532e-2+y*0.32411e-3))))));
266     } else {
267         y=3.75/ax;
268         ans=0.2282967e-1+y*(-0.2895312e-1+y*(0.1787654e-1
269         -y*0.420059e-2));
270         ans=0.39894228+y*(-0.3988024e-1+y*(-0.362018e-2
271         +y*(0.163801e-2+y*(-0.1031555e-1+y*ans))));
272         ans *= (exp(ax)/sqrt(ax));
273     }
274 return x < 0.0 ? -ans : ans;
275 }
276
277 double bessk1(double x)
278 {
279     double bessi1(double x);
280     double y,ans;
281     if (x <= 2.0) {
282     y=x*x/4.0;
283     ans=(log(x/2.0)*bessi1(x))+(1.0/x)*(1.0+y*(0.15443144
284     +y*(-0.67278579+y*(-0.18156897+y*(-0.1919402e-1
285     +y*(-0.110404e-2+y*(-0.4686e-4)))))));
```

```
286     } else {
287         y=2.0/x;
288         ans=(exp(-x)/sqrt(x))*(1.25331414+y*(0.23498619
289         +y*(-0.3655620e-1+y*(0.1504268e-1+y*(-0.780353e-2
290         +y*(0.325614e-2+y*(-0.68245e-3)))))));
291     }
292 return ans;
293 }
294 
295 int ff(double x){
296     int y;
297 if (x-(int)x >= 0.5) y = (int)x + 1;
298 else y = (int)x;
299 
300 return y;
301 }
302 
303 double *vector(long nl, long nh)
304 {
305     double *v;
306 
307     v = (double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(
      double)));
308 
309     return v-nl+NR_END;
310 }
311 
312 double **matrix(long nrl, long nrh, long ncl, long nch)
313 {
314     long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
315     double **m;
316 
317     m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double
      *)));
318     m += NR_END;
319     m -= nrl;
320     m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*
      sizeof(double)));
321     m[nrl] += NR_END;
322     m[nrl] -= ncl;
323     for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;
324 
325     return m;
326 }
327 
328 void free_vector(double *v, long nl, long nh)
329 {
330     free((FREE_ARG) (v+nl-NR_END));
331 }
332 
333 void free_matrix(double **m, long nrl, long nrh, long ncl,
    long nch)
334 {
335     free((FREE_ARG) (m[nrl]+ncl-NR_END));
```

```
336     free((FREE_ARG) (m+nrl-NR_END));
337 }
338
339 void powell(double p[], double **xi, int n, double ftol, int
      *iter, double *fret, double (*func)(double []))
340 {
341     int i,ibig,j;
342     double del,fp,fptt,t,*pt,*ptt,*xit;
343
344     pt=vector(1,n);
345     ptt=vector(1,n);
346     xit=vector(1,n);
347     *fret=(*func)(p);
348     for (j=1;j<=n;j++) pt[j]=p[j];
349     for (*iter=1;;++(*iter)) {
350         fp=(*fret);
351         ibig=0;
352         del = 0.0;
353         for (i=1;i<=n;i++) {
354             for (j=1;j<=n;j++) xit[j]=xi[j][i];
355             fptt=(*fret);
356             linmin(p,xit,n,fret,func);
357             if (fptt-(*fret) > del) {
358                 del=fptt-(*fret);
359                 ibig=i;
360             }
361         }
362         if (2.0*(fp-(*fret)) <= ftol*(fabs(fp)+fabs(*fret))+
              TINY) {
363             free_vector(xit,1,n);
364             free_vector(ptt,1,n);
365             free_vector(pt,1,n);
366             return;
367         }
368         if (*iter == ITMAX) {
369             printf("powell exceeding maximum iterations.");
370             exit(EXIT_FAILURE);
371         }
372         for (j=1;j<=n;j++) {
373             ptt[j]=2.0*p[j]-pt[j];
374             xit[j]=p[j]-pt[j];
375             pt[j]=p[j];
376         }
377         fptt=(*func)(ptt);
378         if (fptt < fp) {
379             t=2.0*(fp-2.0*(*fret)+fptt)*SQR(fp-(*fret)-del)-
                del*SQR(fp-fptt);
380             if (t < 0.0) {
381                 linmin(p,xit,n,fret,func);
382                 for (j=1;j<=n;j++) {
383                     xi[j][ibig]=xi[j][n];
384                     xi[j][n]=xit[j];
385                 }
386             }
```

```
387         }
388     }
389 }
390
391 int ncom;
392 double *pcom, *xicom, (*nrfunc)(double []);
393
394 void linmin(double p[], double xi[], int n, double *fret,
      double (*func)(double []))
395 {
396     int j;
397     double xx,xmin,fx,fb,fa,bx,ax;
398
399     ncom=n;
400     pcom=vector(1,n);
401     xicom=vector(1,n);
402     nrfunc=func;
403     for (j=1;j<=n;j++) {
404         pcom[j]=p[j];
405         xicom[j]=xi[j];
406     }
407     ax=0.0;
408     xx=1.0;
409     mnbrak(&ax,&xx,&bx,&fa,&fx,&fb,f1dim);
410     *fret=brent(ax,xx,bx,f1dim,TOL,&xmin);
411     for (j=1;j<=n;j++) {
412         xi[j] *= xmin;
413         p[j] += xi[j];
414     }
415     free_vector(xicom,1,n);
416     free_vector(pcom,1,n);
417 }
418
419 double f1dim(double x)
420 {
421     int j;
422     double f,*xt;
423
424     xt=vector(1,ncom);
425     for (j=1;j<=ncom;j++) xt[j]=pcom[j]+x*xicom[j];
426     f=(*nrfunc)(xt);
427     free_vector(xt,1,ncom);
428     return f;
429 }
430
431 double brent(double ax, double bx, double cx, double (*f)(
      double), double tol, double *xmin)
432 {
433     int iter;
434     double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm
        ;
435     double e=0.0;
436
437     a=(ax < cx ? ax : cx);
```

```
438     b=(ax > cx ? ax : cx);
439     x=w=v=bx;
440     fw=fv=fx=(*f)(x);
441     for (iter=1;iter<=ITMAX;iter++) {
442         xm=0.5*(a+b);
443         tol2=2.0*(tol1=tol*fabs(x)+ZEPS);
444         if (fabs(x-xm) <= (tol2-0.5*(b-a))) {
445             *xmin=x;
446             return fx;
447         }
448         if (fabs(e) > tol1) {
449             r=(x-w)*(fx-fv);
450             q=(x-v)*(fx-fw);
451             p=(x-v)*q-(x-w)*r;
452             q=2.0*(q-r);
453             if (q > 0.0) p = -p;
454             q=fabs(q);
455             etemp=e;
456             e=d;
457             if (fabs(p) >= fabs(0.5*q*etemp) || p <= q*(a-x)
                  || p >= q*(b-x))
458                 d=CGOLD*(e=(x >= xm ? a-x : b-x));
459             else {
460                 d=p/q;
461                 u=x+d;
462                 if (u-a < tol2 || b-u < tol2)
463                     d=SIGN(tol1,xm-x);
464             }
465         }
466         else {
467             d=CGOLD*(e=(x >= xm ? a-x : b-x));
468         }
469         u=(fabs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
470         fu=(*f)(u);
471         if (fu <= fx) {
472             if (u >= x) a=x; else b=x;
473             SHFT(v,w,x,u)
474             SHFT(fv,fw,fx,fu)
475         }
476         else {
477             if (u < x) a=u; else b=u;
478             if (fu <= fw || w == x) {
479                 v=w;
480                 w=u;
481                 fv=fw;
482                 fw=fu;
483             }
484             else if (fu <= fv || v == x || v == w) {
485                 v=u;
486                 fv=fu;
487             }
488         }
489     }
490     printf("Too many iterations in brent");
```

```
491     exit(EXIT_FAILURE);
492 }
493
494 void mnbrak(double *ax, double *bx, double *cx, double *fa,
      double *fb, double *fc, double (*func)(double))
495 {
496     double ulim,u,r,q,fu,dum;
497
498     *fa=(*func)(*ax);
499     *fb=(*func)(*bx);
500     if (*fb > *fa) {
501         SHFT(dum,*ax,*bx,dum)
502         SHFT(dum,*fb,*fa,dum)
503     }
504     *cx=(*bx)+GOLD*(*bx-*ax);
505     *fc=(*func)(*cx);
506     while (*fb > *fc) {
507         r=(*bx-*ax)*(*fb-*fc);
508         q=(*bx-*cx)*(*fb-*fa);
509         u=(*bx)-((*bx-*cx)*q-(*bx-*ax)*r)/(2.0*SIGN(FMAX(fabs
              (q-r),TINY),q-r));
510         ulim=(*bx)+GLIMIT*(*cx-*bx);
511         if ((*bx-u)*(u-*cx) > 0.0) {
512             fu=(*func)(u);
513             if (fu < *fc) {
514                 *ax=(*bx);
515                 *bx=u;
516                 *fa=(*fb);
517                 *fb=fu;
518                 return;
519             }
520             else if (fu > *fb) {
521                 *cx=u;
522                 *fc=fu;
523                 return;
524             }
525             u=(*cx)+GOLD*(*cx-*bx);
526             fu=(*func)(u);
527         }
528         else if ((*cx-u)*(u-ulim) > 0.0) {
529             fu=(*func)(u);
530             if (fu < *fc) {
531                 SHFT(*bx,*cx,u,*cx+GOLD*(*cx-*bx))
532                 SHFT(*fb,*fc,fu,(*func)(u))
533             }
534         }
535         else if ((u-ulim)*(ulim-*cx) >= 0.0) {
536             u=ulim;
537             fu=(*func)(u);
538         }
539         else {
540             u=(*cx)+GOLD*(*cx-*bx);
541             fu=(*func)(u);
542         }
```

```
543             SHFT(*ax,*bx,*cx,u)
544             SHFT(*fa,*fb,*fc,fu)
545         }
546 }
```

7.2 XploRe Codes

7.2.1 mlgh.xpl

Call dll file to use function "mlgh" in XploRe environment.

```
proc(lambda,alpha,beta,delta,mu,ml)= mlgh(data)
; ------------------------------------------------------
; Macro       mlgh
; ------------------------------------------------------
; Description call dll file to use function "mlgh"
;             in XploRe environment.
;             estimate the five parameters of GH
;             distribution with fractional lambda.
; ------------------------------------------------------
; Usage       {lambda,alpha,beta,delta,mu,ml} = mlgh(data)
; Input
; Parameter   data
; Definition  numeric, data to be estimated
; Output
; Parameter   lambda
; Definition  scalar, estimated value of parameter lambda
; Parameter   alpha
; Definition  scalar, estimated value of parameter alpha
; Parameter   beta
; Definition  scalar, estimated value of parameter beta
; Parameter   delta
; Definition  scalar, estimated value of parameter delta
; Parameter   mu
; Definition  scalar, estimated value of parameter mu
; Parameter   ml
; Definition  scalar, value of log-likelihood
; ------------------------------------------------------

lambda = 0
alpha = 0
delta = 0
beta = 0
mu = 0
ml = 0
length = rows(data)

h = dlopen("Project2.dll")
dlcall(h,"mlgh",data,length,lambda,alpha,beta,delta,mu,ml)
dlclose()
endp
```

7.2.2 mlghint.xpl

Call dll file to use function "mlghint" in XploRe environment.

```
proc(lambda,alpha,beta,delta,mu,ml)= mlghint(data)
; ---------------------------------------------
; Macro       mlghint
; ---------------------------------------------
; Description call dll file to use function "mlghint"
;             in XploRe environment.
;             estimate the five parameters of GH
;             distribution with integer lambda.
; ---------------------------------------------
; Usage       {lambda,alpha,beta,delta,mu,ml} = mlghint(data)
; Input
; Parameter   data
; Definition  numeric, data to be estimated
; Output
; Parameter   lambda
; Definition  scalar, estimated value of parameter lambda
; Parameter   alpha
; Definition  scalar, estimated value of parameter alpha
; Parameter   beta
; Definition  scalar, estimated value of parameter beta
; Parameter   delta
; Definition  scalar, estimated value of parameter delta
; Parameter   mu
; Definition  scalar, estimated value of parameter mu
; Parameter   ml
; Definition  scalar, value of log-likelihood
; ---------------------------------------------

lambda = 0
alpha = 0
delta = 0
beta = 0
mu = 0
ml = 0
length = rows(data)

h = dlopen("Project3.dll")
dlcall(h,"mlghint",data,length,lambda,alpha,beta,delta,mu,ml)
dlclose()
endp
```

7.2.3 *ghmv.xpl*

Calculate the mean and vaiance of a given GH distribution.

```
proc(M,V)=ghmv(l,a,b,d,m)
; ---------------------------------------------
; Library  distribs
; ---------------------------------------------
; Macro       ghmv
; ---------------------------------------------
```

```
; Description   calculate the mean and variance of a given
;               generalized hyperbolic distribution.
; ---------------------------------------------
; Usage       {M,V} = ghmv(l,a,b,d,m)
; Input
; Parameter   l
; Definition  scalar, parameter lambda of the GH distribution
; Parameter   a
; Definition  scalar, parameter alpha of the GH distribution
; Parameter   b
; Definition  scalar, parameter beta of the GH distribution
; Parameter   d
; Definition  scalar, parameter delta of the GH distribution
; Parameter   m
; Definition  scalar, parameter mu of the GH distribution
; Output
; Parameter   M
; Definition  scalar, mean of the GH distribution
; Parameter   V
; Definition  scalar, Variance of the GH distribution
; ---------------------------------------------
; Example       x = ghmv(-0.5,1,0,1,0)
;                           x.M
;                           x.V
; ---------------------------------------------
; Result      Contents of M
;                           [1,]        0
;                           Contents of V
;                           [1,]        1
; ---------------------------------------------
; Author      Congcong Wang
; ---------------------------------------------


g = sqrt(a^2-b^2)

M = m + b*d/g*mbessel3(l+1,d*g)/mbessel3(l,d*g)

V = mbessel3(l+2,d*g)/mbessel3(l,d*g)
V = V - (mbessel3(l+1,d*g)/mbessel3(l,d*g))^2
V = V*b^2/g^2
V = V + mbessel3(l+1,d*g)/(d*g*mbessel3(l,d*g))
V = V*d^2

endp
```

BIBLIOGRAPHY

Atkinson, A. C. (1982). The simulation of generalized inverse gaussian and hyperbolic random variables, *SIAM Journal of Scientific and Statistical Computations* **3**: 503–515.

Barndorff-Nielsen, O. E. (1977). Exponentially decreasing distributions for the logarithm of particle size, *Proceedings of the Royal Society of London* **A 353**: 401–419.

Barndorff-Nielsen, O. E. (1978). Hyperbolic distributions and distributions on hyperbolae, *Scandinavian Journal of Statistics* **5**: 151–157.

Barndorff-Nielsen, O. E. (1982). The hyperbolic distribution in statistical physics, *Scandinavian Journal of Statistics* **9**: 43–46.

Barndorff-Nielsen, O. E. (1997). Normal inverse gaussian distributions and stochastic volatility modelling, *Scandinavian Journal of Statistics* **24**: 1–13.

Barndorff-Nielsen, O. E. (1998). Probability and statistics; selfdecomposability, finance and turbulence, in l. accardi & c. c. heyde (eds), *Proceedings of the conference "Probability towards 2000" held at Columbia University, New York, 2-6 October 1995* **49**: 47–57.

Barndorff-Nielsen, O. E. and Christiansen, C. (1988). Erosion, deposition and size distribution of sand, *Proceedings of the Royal Society London* **A 417**: 335–352.

Barndorff-Nielsen, O. E. and Shephard, N. (2001). Non-gaussian ornstein-uhlenbeck-based models and some of their uses in financial economics (with discussion), *Journal of the Royal Statistical Society Series B* **63**: 167–241.

Barndorff-Nielsen, O. E. and Stelzer, R. (2005). Absolute moments of generalized hyperbolic distributions and approximate scaling of normal inverse gaussian lévy processes, *Working Paper, University of Århus, Munich University of Technology*.

Barndorff-Nielsen, O. E., Blæsild, P. and Schmiegel, J. (2004). A parsimonious and universal description of turbulent velocity increments, *The European Physical Journal B* **41**: 345–363.

Barndorff-Nielsen, O. E., Blæsild, P., Jensen, J. L. and Sørensen, M. (1983). The fascination of sand - with three appendices by r.a.bagnold, *Department of Theoretical Statistics, Institute of Mathematics, University of Århus, Denmark.*

Barndorff-Nielsen, O. E., Blæsild, P., Jensen, J. L. and Sørensen, M. (1985). The fascination of sand, in a. c. atkinson & s. e. fienberg (eds), *A celebration of statistics - The ISI centenary volume, Springer, New York* pp. 57–87.

Barndorff-Nielsen, O. E., Jensen, J. L. and Sørensen, M. (1989). Wind shear and hyperbolic distributions, *Boundary-Layer Meteorology* **49**: 417–431.

Blæsild, P. (1981). The two-dimensional hyperbolic distribution and related distributions, with an application to Johannsen's bean data, *Biometrika* **68**: 251–263.

Blæsild, P. (1999). Generalized hyperbolic and generalized inverse gaussian distributions., *Working Paper, University of Århus.*

Eberlein, E. and Keller, U. (1995). Hyperbolic distributions in finance, *Bernoulli* **1**: 281–299.

Eriksson, A., Forsberg, L. and Ghysels, E. (2004). Approximating the probability distribution of functions of random variables: A new approach, *Série Scientifique 21.*

Franke, J., Härdle, W. and Hafner, C. (2004). *Einführung in die Statistik der Finanzmärkte*, Springer, Berlin.

Gut, A. (1995). *An Intermediate Course in Probability*, Springer, New York.

Hartmann, D. and Bowman, D. (1993). Efficiency of the log-hyperbolic distribution - a case study: Pattern of sediment sorting in a small tidal-inlet - het zwin, the netherlands, *Journal of Coastal Research* **9**: 1044–1053.

Kristjansson, L. and McDougall, I. (1982). Some aspects of the late tertiary geomagnetic field in iceland, *The Geophysical Journal of the Royal Astronomical Society* **68**: 273–294.

Pareto, V. (1896). Cours d'économie politique professé à l'université de Lausanne.

Prause, K. (1997). Modelling financial data using generalized hyperbolic distribtions, *FDM Preprint 48, Department of Mathematical Stochastics, University of Freiburg*.

Prause, K. (1999). The generalized hyperbolic model: Estimation, financial derivatives and risk measures, *Dissertation, Mathematische Fakultät, Albert-Ludwigs-Universität Freiburg im Breisgau*.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2002). *Numerical Recipes in C++*, Cambridge University Press.

Sutherland, R. A. and Lee, C.-T. (1994). Application of the log-hyperbolic distribution to hawai'ian beach sands, *Journal of Coastal Research* **10**: 251–262.

Xu, T. H., Durst, F. and Tropea, C. (1993). The three-parameter log-hyperbolic distribution and its application to particle sizing, *Atomization and Sprays* **3**: 109–124.